



INTERNATIONAL TELECOMMUNICATION UNION

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

T.107

(08/95)

TERMINALS FOR TELEMATIC SERVICES

**ENHANCED MAN MACHINE INTERFACE
FOR VIDEOTEX AND OTHER RETRIEVAL
SERVICES (VEMMI)**

ITU-T Recommendation T.107

(Previously "CCITT Recommendation")

FOREWORD

The ITU-T (Telecommunication Standardization Sector) is a permanent organ of the International Telecommunication Union (ITU). The ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Conference (WTSC), which meets every four years, establishes the topics for study by the ITU-T Study Groups which, in their turn, produce Recommendations on these topics.

The approval of Recommendations by the Members of the ITU-T is covered by the procedure laid down in WTSC Resolution No. 1 (Helsinki, March 1-12, 1993).

ITU-T Recommendation T.107 was prepared by ITU-T Study Group 8 (1993-1996) and was approved under the WTSC Resolution No. 1 procedure on the 11th of August 1995.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

© ITU 1996

All rights reserved. No part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the ITU.

CONTENTS

	<i>Page</i>
1 Scope.....	1
2 References.....	1
3 Definitions and abbreviations.....	2
3.1 Definitions.....	2
3.2 Abbreviations.....	3
4 General model.....	4
4.1 Introduction.....	4
4.2 Definition of the VEMMI elements.....	4
4.2.1 VEMMI object definition and identification.....	5
4.2.2 VEMMI component definition.....	5
4.2.3 VEMMI component item definition.....	6
4.2.4 Resource definition.....	6
4.2.5 Transfer of VEMMI objects.....	6
4.2.6 Resource transfer.....	6
4.3 VEMMI plane structure model.....	7
4.3.1 The standard plane for videotex data.....	7
4.3.2 The VEMMI objects plane.....	7
4.4 Operation modes for VEMMI terminals.....	7
4.4.1 The standard mode.....	7
4.4.2 The VEMMI mode.....	7
4.4.3 Switching between standard mode and VEMMI mode.....	9
4.5 VEMMI elements data content.....	9
4.5.1 Text data definition.....	9
4.5.2 Bitmaps.....	10
4.5.3 Colour.....	11
4.5.4 Videotex data.....	11
4.5.5 Sound.....	11
4.5.6 Graphical data.....	11
4.5.7 Video data.....	11
4.6 VEMMI objects positioning and dimensioning.....	11
4.6.1 Positioning.....	11
4.6.2 Dimensioning.....	12
4.7 VEMMI elements states and state parameters.....	13
4.7.1 Object.....	13
4.7.2 Component.....	16
4.8 Local action management.....	17
4.9 Memory considerations.....	18
4.10 Common rules for object handling.....	18
4.10.1 Active state and focus management.....	18
4.10.2 Behaviour of the modal mode.....	19
4.10.3 Size considerations and clipping.....	19
4.11 Local object storage.....	19
4.12 Symbolic directory names.....	20
4.13 Specific rules for dedicated terminals.....	20
5 Service description.....	21
5.1 Service elements initiated by the VEMMI application and the terminal.....	23
5.2 Service elements initiated by the VEMMI application.....	23
5.2.1 VEMMI_Open.....	23
5.2.2 VEMMI_Close.....	24
5.2.3 VEMMI_Resume.....	24

5.2.4	VEMMI_Suspend	24
5.2.5	VEMMI_Identify_Term_Cap	25
5.2.6	VEMMI_Set_Options	25
5.2.7	VEMMI_Create_Object.....	26
5.2.8	VEMMI_Open_Object	26
5.2.9	VEMMI_Open_Blocking_Object.....	26
5.2.10	VEMMI_Close_Object	27
5.2.11	VEMMI_Close_All.....	27
5.2.12	VEMMI_Destroy_Object.....	27
5.2.13	VEMMI_Obj_Access_Disable	27
5.2.14	VEMMI_Obj_Access_Enable	28
5.2.15	VEMMI_Modify_Component	28
5.2.16	VEMMI_Obj_Location_Change	29
5.2.17	VEMMI_Load_Col_Table.....	29
5.2.18	VEMMI_Reset_Col_Table	30
5.2.19	VEMMI_Open_Application	31
5.2.20	VEMMI_Delete_Outdated_Objects.....	31
5.2.21	VEMMI_Store_Objects	32
5.2.22	VEMMI_Erase_Objects.....	32
5.2.23	VEMMI_User_Lock.....	32
5.2.24	VEMMI_User_Unlock	32
5.2.25	VEMMI_Resource_Transfer	33
5.3	Service elements initiated by the terminal	34
5.3.1	VEMMI_Identify_Term_Cap_Resp	34
5.3.2	VEMMI_Object_Retransmission.....	34
5.3.3	VEMMI_User_Data.....	35
5.3.4	VEMMI_Open_Application_Resp	36
5.3.5	VEMMI_Store_Objects_Resp	36
5.3.6	VEMMI_Error	36
5.3.7	VEMMI_Resource_Transfer_Abort	37
6	VEMMI objects introduction	37
6.1	The application bar	38
6.1.1	Composition.....	38
6.2	The button bar.....	38
6.2.1	Composition.....	38
6.3	The pop-up menu.....	38
6.3.1	Composition.....	38
6.4	The dialogue box	39
6.4.1	Composition.....	39
6.5	Operative object.....	40
6.6	Bitmap resource object	41
6.7	Videotex resource object	41
6.8	Text resource object.....	41
6.9	Font resource object.....	41
6.10	Metacode object.....	41
6.11	The message box.....	41
7	Functional description	41
7.1	General rules for the behaviour of elements	41
7.1.1	User interaction	41
7.1.2	Local actions and reports	41
7.1.3	Relationship between objects and components.....	42
7.1.4	Open/close of audio, video, resource and metacode objects.....	43
7.1.5	Maximize operation	43
7.1.6	Notational Conventions	44
7.1.7	Mnemonic	44

	<i>Page</i>	
7.2	Text formats.....	44
7.2.1	VEMMI high quality text.....	44
7.2.2	Text labels and titles	45
7.3	The Application Bar	46
7.3.1	Composition.....	48
7.4	The Button Bar	53
7.4.1	Composition.....	54
7.5	The Pop-Up Menu	55
7.5.1	Composition.....	57
7.6	The Dialogue Box.....	58
7.6.1	Composition.....	61
7.7	The Message Box	83
7.8	Operative object.....	85
7.9	Bitmap resource object	87
7.10	Videotex resource object	88
7.11	Text resource object.....	88
7.12	Font resource object.....	89
7.13	Metacode object.....	89
7.14	VEMMI bitmap data type definition.....	89
7.15	The VEMMI content encoding identification catalogue.....	89
8	Complete coded representation of the VEMMI	91
8.1	Introduction	91
8.2	Notation used.....	91
8.3	Overall switching of coding environment	91
8.3.1	Switching into the VEMMI mode.....	91
8.3.2	ISO/IEC 9281 [14] syntax structure.....	93
8.4	VEMMI Command Syntax.....	95
8.5	Objects, components.....	98
8.6	Local actions.....	102
9	Encoding	103
9.1	Command structure.....	103
9.2	Object, component and attribute structure	103
9.3	Terminal symbols encoding.....	104
9.3.1	Opcodes	104
9.3.2	Integers	104
9.3.3	Enumerated	105
9.3.4	Strings	105
9.3.5	NDC	106
9.4	Attributes and lower level symbols.....	107
9.5	Opcodes	108
9.6	Syntax of the VEMMI_Modify_Component.....	111
9.7	Defaults.....	113
10	Introduction of the VEMMI service into existing Videotex Recommendations.....	116
10.1	Introduction of the VEMMI to T.101 [4]	116
10.2	Introduction of the VEMMI to T.105 [6]	116
Annex A	– T.51String	116
A.1	Scope	116
A.2	Graphic character sets.....	116
A.3	Code extension technique	119
A.4	Repertoire of the Latin based character set.....	119
A.5	Control functions	119
Annex B	– Mandatory subset of ISO 8859 [13]	120
Annex C	– Minimum datatype kernel	121

SUMMARY

This Recommendation specifies the syntax to be used by videotex or other retrieval services for implementation of an enhanced man machine interface (VEMMI).

The VEMMI is a means to improve the ergonomics and interactivity of retrieval systems using graphical dialogue elements such as application bar, button bar, pop-up menus, dialogue box.

The standard plane continues to receive the standard data of the underlying platform. Standard Videotex Applications can therefore run also on a VEMMI terminal. The VEMMI objects plane receives VEMMI objects. An automatic switching mechanism between standard mode and VEMMI mode is provided.

This Recommendation defines datatypes (VEMMI-high quality text, VEMMI bitmaps) and allows the integration of other widely used datatypes (e.g. JPEG, BMP, WAVE, MIDI, Videotex) in the VEMMI dialogue. Also operative objects are defined to extend the capabilities of a VEMMI application during runtime.

VEMMI objects can be distributed between a host and a VEMMI terminal to improve the performance of interactive applications. The VEMMI application can control (save, load, update) VEMMI objects on the terminal.

This Recommendation contains the service description, the service elements and their coding.

ENHANCED MAN MACHINE INTERFACE FOR VIDEOTEX AND OTHER RETRIEVAL SERVICES (VEMMI)

(Geneva, 1995)

1 Scope

This Recommendation specifies the data syntax to be used by Videotex and Multimedia/Hypermedia Information retrieval services for implementation of the Videotex Enhanced Man Machine Interface (VEMMI).

In the Videotex case, this Recommendation is applicable to both the Videotex service and the attached Videotex terminals. Those terminals may be connected to the Videotex service via the Public Switched Telephone Network (PSTN), Integrated Services Digital Network (ISDN) or Packet Switched Public Data Network (PSPDN). Typically, the terminals should support ISDN Syntax-Based Videotex (SBV).

This Recommendation can also be used for any kind of retrieval service (not related to Videotex) by using the relevant underlying platform and content data types.

2 References

The following Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; all users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published.

- [1] CCITT Recommendation T.50 (1992), *International Reference Alphabet (IRA) (Formerly International Alphabet No. 5 or IA5) – Information technology – 7-bit coded character set for information interchange.*
- [2] CCITT Recommendation T.51 (1992), *Latin based coded character sets for telematic services.*
- [3] ITU-T Recommendation T.52 (1993), *Non-Latin coded character sets for telematic services.*
- [4] ITU-T Recommendation T.101 (1994), *International interworking for videotex services.*
- [5] ITU-T Recommendation T.102 (1993), *Syntax-based videotex end-to-end protocols for circuit mode ISDN.*
- [6] ITU-T Recommendation T.105 (1994), *Syntax-based videotex application layer protocol.*
- [7] ITU-T Recommendation H.261 (1993), *Video codec for audiovisual services at $p \times 64$ kbit/s.*
- [8] ITU-T Recommendation H.320 (1993), *Narrow-band visual telephone systems and terminal equipment.*
- [9] ITU-T Recommendation F.300 (1993), *Videotex service.*
- [10] ISO 2022:1986, *Information Processing – ISO 7-bit and 8-bit coded character sets – Code extension techniques.*
- [11] ISO 2375:1985, *Data processing – Procedure for registration of escape sequences.*
- [12] ISO 8632:1992, *Information technology – Computer graphics – Metafile for storage and transfer of picture description information.*
- [13] ISO 8859:1987, *Information Processing - 8-bit single-byte coded graphic character sets.*
- [14] ISO 9281:1990, *Information technology – Picture coding methods.*
- [15] ISO 10646-1:1993, *Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Part 1: Architecture and basic multilingual plane.*

- [16] ISO-IS 10918-1:1994, *Digital compression and coding of continuous-tone still images. Part 1: Requirements and guidelines.*
- [17] ISO 11172-1:1993, *Information technology – Coding of moving pictures and associated audio for digital storage media up to about 1.5 Mbit/s - Part 1: Systems.*
- [18] ISO 11172-2:1993, *Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s – Part 2. Video.*

3 Definitions and abbreviations

3.1 Definitions

For the purposes of this Recommendation, the following definitions apply:

- 3.1.1 controls:** Visual user-interface elements that allow a user to interact with data.
- 3.1.2 Defined Display Area (DDA):** See Recommendation F.300 [9].
- 3.1.3 emphasis:** Highlighting, colour change or other visible indication of the condition of an element or choice and the effect of that condition on a user's ability to interact with that element. Emphasis can also give additional information about the state of an object. The method used to emphasize an element is terminal dependent.
- 3.1.4 label:** Text data associated with a VEMMI component to inform the user of the purpose of a particular component or item.
- 3.1.5 local manager:** See VEMMI local manager.
- 3.1.6 mnemonic:** A single, easy-to-remember alphanumeric character that activates a VEMMI Menu Choice component and validates it. A Mnemonic character can also be used to validate an active Push Button.
- 3.1.7 modal mode:** When a VEMMI object is "modal" it is not possible for the user to leave this VEMMI object to the benefit of another VEMMI object of the same application with the different possible access tools. Each attempt to access another object by the user is refused and possibly indicated by a sound signal.
- 3.1.8 resource transfer:** Mechanism to transfer files referenced by VEMMI resource objects from a VEMMI application to a VEMMI terminal.
- 3.1.9 stretched presentation:** Shrinked or enlarged display of a bitmap in order to meet given space requirements.
- 3.1.10 tiled presentation:** Repeated display of a given bitmap in horizontal and/or vertical direction in order to meet given space requirements.
- 3.1.11 videotex application:** Videotex application using encoded data, protocols and profiles, as defined in the Videotex Recommendations referred to in clause 2. A Videotex application does not use a VEMMI service, data and protocols (see Recommendation F.300 [9]).
- 3.1.12 videotex data:** Data interchanged between a Videotex application and a Videotex terminal.
- 3.1.13 validation:** User activation action followed by a confirmation of the choice with a keyboard or with a pointing device.
- 3.1.14 VEMMI application:** Application offering an enhanced man machine interface as described in this Recommendation.
- 3.1.15 VEMMI data:** VEMMI objects description and contents and VEMMI commands exchanged between the VEMMI application and the VEMMI terminal.
- 3.1.16 VEMMI local manager:** Software running in the VEMMI terminal to handle and to present the VEMMI objects that are sent to the user by the VEMMI application.
- 3.1.17 VEMMI terminal:** Terminal which is able to run a VEMMI local manager.
- 3.1.18 videotex host computer:** See Recommendation F.300 [9].
- 3.1.19 videotex terminal:** See Recommendation F.300 [9].

3.2 Abbreviations

For the purposes of this Recommendation, the following abbreviations apply:

BIN	Bitmap Identification Number
BMP	Microsoft Windows Device-Independent Bitmap
CCITT	The International Telegraph and Telephone Consultative Committee
CD-ROM	Compact Disk-Read Only Memory
CGM	Computer Graphics Metafile
CIN	Component Identification Number
CMI	Coding Method Identifier
CR	Carriage Return
DDA	Defined Display Area
DIB	Device-Independent Bitmap
DRCS	Dynamically Redefinable Character Set
DS I	Data Syntax according to Annex B/T.101 [4]
DS II	Data Syntax according to Annex C/T.101 [4]
DS III	Data Syntax according to Annex D/T.101 [4]
ESC	Escape
ETS	European Telecommunication Standard
ETSI	European Telecommunications Standards Institute
FIN	Font Identification Number
G0	Primary character set of Recommendation T.51 [2]
G2	Supplementary character set of Recommendation T.51 [2]
GIF	Graphics Interchange Format
GMT	Greenwich Mean Time
GUI	Graphical User Interface
IEC	International Electrotechnical Commission
IRV	International Reference Version
IS	International Standard
ISDN	Integrated Services Digital Network
ISO	International Organization For Standardization
ITU-T	International Telecommunication Union – Telecommunication Standardization
JIS	Japanese Institute for Standardization
JPEG	Joint Photographic Experts Groups
LF	Line Feed
LI	Length Indicator
MDI	More Data Indicator
MIDI	Musical Instrument Digital Interface
MPEG	Moving Picture Experts Group
NDC	Normalized Device Coordinate

OIN	Object Identification Number
PCD	Picture Code Delimiter
PCE	Picture Control Entity
PDE	Picture Data Entity
PE	Picture Element
PI	Picture Identifier
PM	Picture Mode
PSPDN	Packet Switched Public Data Network
PSTN	Packet Switched Telephone Network
RGB	Red Green Blue
SBV	Syntax-Based Videotex
TE	Terminal Equipment
TFI	Terminal Facility Identifier
TIN	Text Identification Number
TLV	Type Length Value
TV	Television
UI	User Interface
VEMMI	Videotex Enhanced Man Machine Interface
VIN	Videotex Identification Number
VPDE	Videotex Presentation Data Element
VTX	Videotex

4 General model

4.1 Introduction

Between a host and a VEMMI terminal, a VEMMI service handles:

- general VEMMI objects as described in this Recommendation;
- data contents as defined in this Recommendation;
- data contents as referred to in this Recommendation.

A VEMMI terminal may also handle a Videotex application using encoded data and protocols as described in the Videotex Recommendations referred to in clause 2.

4.2 Definition of the VEMMI elements

The logical units which form the structure of the VEMMI shall be named and defined as follows:

- VEMMI objects or objects;
- VEMMI components or components;
- VEMMI component item or items.

VEMMI element is a generic name used in this Recommendation to designate an object, a component or an item.

An example is given Figure 1.

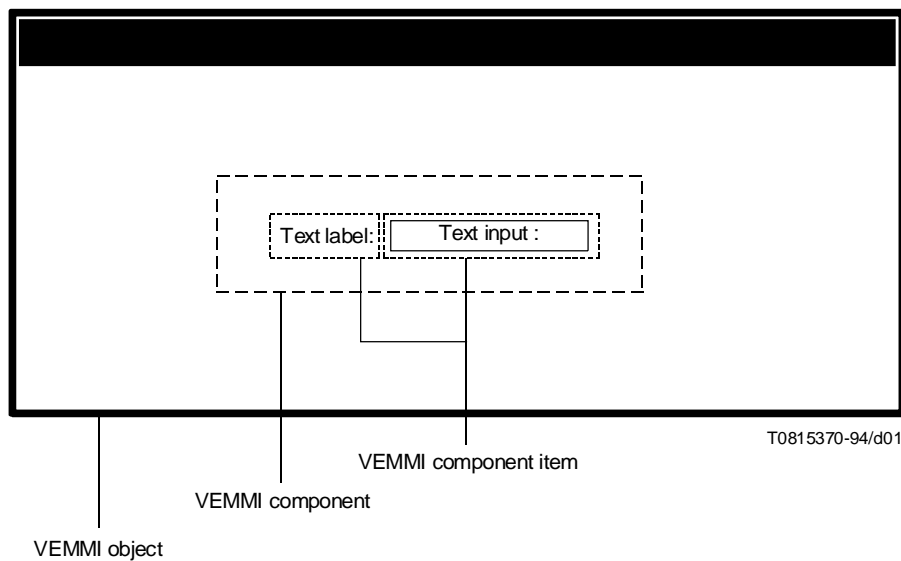


FIGURE 1/T.107
Example showing objects/components/items

4.2.1 VEMMI object definition and identification

The following four different types of VEMMI objects are defined in this Recommendation:

- display objects;
- operative objects;
- resource objects;
- metacode objects.

If not stated otherwise, the term object used alone always refers to a display object.

VEMMI objects are the logical units which are used by a VEMMI application to interact with the user.

VEMMI objects are composed of different components.

The objects are only defined regarding their functionality, their size and position relative to the Defined Display Area (DDA). The representation of the objects is terminal dependent.

Every object shall be identified by an Object Identification Number (OIN) which shall be unique within a VEMMI application at any one time.

4.2.2 VEMMI component definition

VEMMI components always belong to a VEMMI object and are only valid within this object. The object, to which a component belongs, is named parent object.

In order to transport information, components may carry a data content (see 4.5).

The components are only defined regarding their functionality, their type of content and their size and position relative to the object. The representation of the components is terminal dependent. The representation of data content is specified either by this Recommendation (for the datatypes that are defined within this Recommendation) or by the corresponding presentation standard (for datatypes that are defined outside this Recommendation).

Every component shall be identified by a Component Identification Number (CIN) which shall be unique within an object.

4.2.3 VEMMI component item definition

The subunit of a VEMMI component is a component item. Every item is an integral part of a component. The definition of a component item is only valid within this component.

4.2.4 Resource definition

Resources are elements which can be referenced by components or objects. One resource can be referenced by more than one element. The following resources are defined:

- the colour table which is unique in one application;
- files stored in the terminal (identified by filenames) can contain sound data, operative objects etc.;
- a combination of a font and a set of attributes is a resource object. It is identified via a Font Identification Number (FIN);
- a bitmap is a resource object. It is identified via a Bitmap Identification Number (BIN);
- text can be a resource object. It is identified via a Text Identification Number (TIN);
- Videotex can be a resource object. It is identified via a Videotex Identification Number (VIN);
- objects sets stored in the terminal between two sessions (they are identified via attributes).

NOTE – FIN, BIN, VIN are Object Identification Numbers (OINs). The terms FIN, BIN and VIN are only used to clearly indicate that the corresponding object is a resource object.

4.2.5 Transfer of VEMMI objects

VEMMI objects can be transmitted to the VEMMI terminal using a telecommunication network. If they are stored then in the terminal, they become local objects. VEMMI objects can also be downloaded using any filetransfer. They become local objects as well and are treated in the same way as objects transferred within the VEMMI dialogue and stored by the corresponding service primitive. VEMMI objects may also be transferred to the VEMMI terminal by postal mail (CD-ROM, diskettes, etc.).

4.2.6 Resource transfer

VEMMI resource objects can reference files that contain the resource display data. These files are called resource data files.

VEMMI specifies the way the resource data files are transmitted to the terminal (VEMMI resource transfer). VEMMI resources data files may also be transferred to the VEMMI terminal using the file transfer used in the standard service in which the terminal operates or by postal mail (CD-ROM, diskettes, etc.).

In order to provide a satisfactory level of quality and features matching the VEMMI functionality, the resource transfer mechanism offers the following facilities:

- the resource transfer can be performed without the user being aware of it;
- the resource transfer can be performed as a parallel task. During the resource transfer, the user may continue to interact with the VEMMI application (although if the network speed is not sufficient, the server response time may be adversely affected by the resource transfer operation);
- the application may open a VEMMI object (window) displaying the actual status of the transfer (e.g. a graphics representing the percentage of the resource copied), and optionally offer a method to cancel, hold and resume the transfer (e.g. using buttons);
- several resource transfers can be performed independently;
- possible user interaction on the resource transfer operation (abort).

The support of resources is optional for a VEMMI terminal. If the terminal supports the local storage facility, it shall support the VEMMI resource transfer as well.

4.3 VEMMI plane structure model

The VEMMI display model consists of two independent planes:

- the standard data memory and standard data window;
- the VEMMI data memory and VEMMI data window.

This model and a possible terminal structure is presented in Figure 2.

A VEMMI terminal shall implement the behaviour of this display model. However, no assumption is made on the real physical plane structure of the terminal and how the terminal implements that plane structure model.

The standard plane is the output area where the retrieval service used as a platform for the VEMMI service displays its data. If a Videotex service is used as a platform, the standard plane is the DDA that is used by a regular Videotex application. In the following subclauses, it is assumed that Videotex is used as a platform. However, all the rules apply as well if the underlying platform is different from Videotex (e.g. VT100, TV-Channel).

The two output planes are independent with respect to their positions and dimensions on the terminal screen.

4.3.1 The standard plane for videotex data

The standard plane receives standard Videotex data. It shall continue to support the Videotex data rules and priorities defined in the Videotex Recommendations referred to in clause 2.

4.3.2 The VEMMI objects plane

This plane receives VEMMI objects described in this Recommendation and their encoded data contents. The VEMMI objects plane shall be able to handle object overlapping and restoring mechanisms as referred to in 4.6 and 4.10.1. Within this plane, and for a given data content, the data rules and priorities as defined in the corresponding standards apply.

4.4 Operation modes for VEMMI terminals

Regardless of the current operation mode of the terminal, it shall always provide specific tools to manage the platform specific functions (disconnect, etc.).

4.4.1 The standard mode

This is the initial mode of operation of a VEMMI terminal when powered on or reset.

In standard mode, the VEMMI terminal displays standard data in the standard plane. This standard datatype depends on the retrieval service to which the VEMMI terminal is connected. Standard data are fully visible. User inputs are not controlled by the VEMMI local manager.

The terminal shall ignore all VEMMI commands except VEMMI_Open or VEMMI_Resume if the VEMMI session previously was suspended with a VEMMI_Suspend. This command shall perform a switch to the VEMMI mode (VEMMI On/Off switches are then switched to the On position).

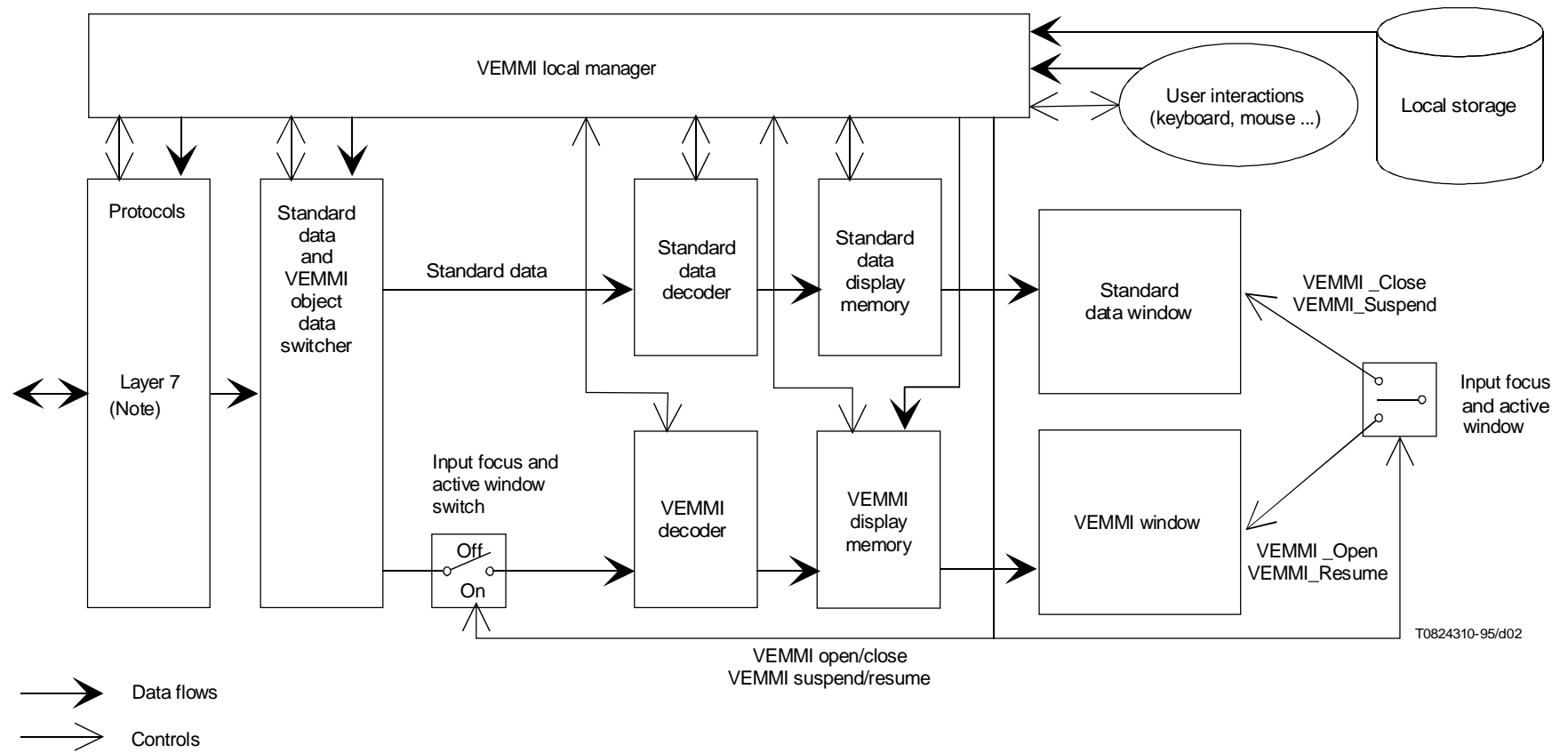
4.4.2 The VEMMI mode

In VEMMI mode, a VEMMI terminal can receive standard data and VEMMI objects data in two parallel paths, as shown in Figure 2.

VEMMI object data are displayed in the VEMMI objects plane. Standard data are displayed in the standard plane. The display order for objects and for components within objects shall correspond to the ascending order of their CINs (component with the smallest CIN is displayed first).

In VEMMI mode, the received standard data is used to update the standard data display memory. An update of the standard data window does not cause its activation.

User inputs are controlled by the VEMMI local manager on the VEMMI window, which is the active window in this mode.



NOTE – Standard protocols including Syntax-Based Videotex (SBV) based protocols or protocols amended for VEMMI mode as defined in clause 5.

FIGURE 2/T.107
Example of possible VEMMI terminal structure

4.4.3 Switching between standard mode and VEMMI mode

From the standard mode, a VEMMI terminal shall be switched to the VEMMI mode using the VEMMI_Open or VEMMI_Resume command.

From the initial mode the terminal can be switched to the VEMMI mode using the VEMMI_Open command.

When a VEMMI_Close command is received, the VEMMI terminal shall switch in the standard operation mode.

The VEMMI_Resume and the VEMMI_Suspend command can be used to temporarily switch between the two modes.

4.5 VEMMI elements data content

The following content datatypes and attributes are defined within this Recommendation:

- a) VEMMI high quality text;
- b) VEMMI bitmaps;
- c) colour.

The following content datatypes and attributes are defined outside this Recommendation, but used as data content for VEMMI elements in this Recommendation:

- a) Videotex data (as optional data content of the Graphic Output Area);
- b) bitmaps (JPEG, GIF, BMP);
- c) sound (WAVE, MIDI);
- d) graphical data;
- e) video data.

It is mandatory for a VEMMI terminal to understand all commands related to the above mentioned data contents.

No specific content datatype is mandatory to be used in a VEMMI application. However, in order to provide a satisfactory VEMMI service, a terminal should, as a minimum requirement, support the datatypes listed in Annex C.

4.5.1 Text data definition

Text Data are encoded in accordance with the ISO 8859-Series [13]. Depending on the terminal implementation, only a part of 8859-Series may be supported (see Annex B for 8859 mandatory subset of ISO 8859-Series). In addition, the host application can use the VEMMI_Identify_Term_Cap command to enquire about the character sets supported in the terminal. A specific character set can be selected using the VEMMI_Set_Options command. This text data encoding is considered as the basic text encoding. When several character sets are used in the same application (multilingual application) the application can use the registered ISO appropriate designation escape sequences, as defined in ISO 2022 [10] and ISO 8859-Series [13] to switch between the different character sets.

The following text data encodings can be used optionally:

- ISO 10646-1 Unicode [15];
- Recommendation T.51 [2] as given in Annex A;
- Recommendation T.52 [3].

For displaying window titles, menu choices and labels, the terminal application can use the local system font. For the application specific text-output, VEMMI offers enhanced capabilities. However, simple text based terminals can use the system font to display the high quality text. These terminals may then ignore the text attributes that they are not able to process and display text using the system font, in the available system colour, on the system background.

Text data can be part of an object or component definition or it can be defined as a resource object. The following text attributes are specified:

- text colour;
- text font;
- text height.

The **text colour** defines the colour of the characters foreground. The character background is terminal specific. The **text font** is a set of characters with a particular, similar character design. In this context, an application can choose a font family, the specific font pertaining to this family is selected by the terminal application (in this Recommendation, by “font” a font family is meant). Table 1 presents the VEMMI font families:

TABLE 1/T.107
VEMMI font families

Font family	Font characteristics	Example
SWISS	Proportional, without serifs	Helvetica, Switzerland
ROMAN	Proportional, with serifs	Times roman
FIXFONT	Monospaced	Courier

The fonts of these families should support variable sizes (scalable fonts).

The **text height** specifies the height of the font. It is the height of the character cell, including a possible internal leading. The local manager shall insert additional space between adjacent text rows. The height is measured in points (1/72 inch). One point corresponds to 1/400 NDC. Italicized characters can be achieved with the attribute **Italic**. **Bold** increases the line thickness and **Underline** draws a line under each character.

Example

SWISS, 10 points and bold

Roman, 8 points and underlined

Roman, 16 points, italic

FIXFONT, 12 points, white

A combination of a font and a set of attributes is further named an attributed font and handled as a font resource object. It is referenced via the FIN (Font Identification Number).

The following control characters should be used to format the text correctly:

- SP, NBSP, SHY: see ISO 8859-1 [13];
- CR, LF: see Recommendation T.50 [1].

To provide the “line feed” functionality within the text content of a component or a component item, the Carriage Return (CR) + Line Feed (LF) control characters given in Recommendation T.50 [1] shall be used.

4.5.2 Bitmaps

A bitmap is a pixel matrix containing either colour indices which point into the colour table, or Red Green Blue (RGB)-components. A bitmap can be referenced by an object or component and when it is instantiated by the terminals it appears as a rectangle with a colour pattern of the corresponding matrix of pixels. The definition is almost device-independent while the relationship between the bitmap bits and the pixels on the device is device specific.

The bitmap is handled as a resource object and referenced by objects or components via the “Bitmap Identification Number” (BIN) which is unique in one application. Several bitmaps can be defined at a given point in time.

4.5.3 Colour

The colour table provides a method for accessing the colour capabilities of the terminal device. It is assumed that the device can display at least 256 colours simultaneously. Because the device colour table is shared by more than one application, the UIs provide mechanisms (e.g. logical colour palettes) to support each application with its own table. This subclause defines the colour table for the VEMMI application.

From the 256 colours, the User Interface (UI) reserves 20 colours for a system table. Therefore, the maximum number of colours for the VEMMI colour table is 236. In practice an application will use only a few numbers from these colours. The host application has the possibility to define the colour table by sending the RGB components. Objects, components or bitmaps can select these colours via indices. By default, 16 colour entries (0 to 15) are predefined. If the application uses a colour index > 15 which was not defined, the displayed colour is terminal dependent.

Colour entries may be in use by other active applications in the terminals. This has to be managed by the local UI.

Monochrome terminals support either two colour indices (e.g. black and white or the colour table loaded with shades of one colour).

4.5.4 Videotex data

Data encoded in accordance with Annex B/T.101, Annex C/T.101 and Annex D/T.101 [4], for example: text, geometric.

Data encoded in accordance with Annex E/T.101 [4], for audio data.

Data encoded in accordance with Annex F/T.101 [4], for photographic data.

The support of Videotex data is optional for a VEMMI terminal.

The host application can inquire about the support of these data types using the VEMMI_Identify_Term_Cap command.

4.5.5 Sound

An application can embed references to sound sequences in objects. The terminal stores the sound in files. Upon a user action the sound is processed, possibly with an audio interface card.

The two sound formats are suitable for:

- wave devices (WAVE);
- Musical Instrument Digital Interface devices (MIDI).

Sound can also be provided using the Videotex Audio syntax defined as Videotex data content (see 4.5.4.).

4.5.6 Graphical data

Graphical data, such as lines, arcs, fill areas and the relevant drawing attributes are not directly supported by this Recommendation. However, an application can use VEMMI-integrated programmes (via operative objects), in which graphical drawing operations are performed.

4.5.7 Video data

This Recommendation does not describe video data formats. But it provides the container (object) which can be used to embed moving video in VEMMI applications.

4.6 VEMMI objects positioning and dimensioning

4.6.1 Positioning

The standard plane shall continue to support the coordinate system used at the underlying platform.

The VEMMI objects plane shall support a Normalized Device Coordinate (NDC) system for positioning VEMMI objects and components within the DDA (see Figure 3). The normalized coordinates are theoretically expressed in the range 0;0 to 1;1. The 0;0 coordinate origin reference point represents the upper left-hand corner of the DDA, the 1;1 coordinate point represents the lower right-hand corner of a virtual square DDA with a side equal to the unit. For a non-square DDA, the unit is equal to the greatest side of the DDA.

For typical display devices, using common 4:3 aspect ratio, the horizontal positioning is in the range 0 to 1 corresponding to the whole width of the DDA, the vertical positioning is in the range 0 to 0.75 corresponding to the whole height of the DDA.

The origin reference point to position an object within the DDA is always the upper left corner of the DDA. The origin reference point to position a component within its parent object (container object) is also the upper left corner of a virtual DDA attached to the upper left corner of the object; that virtual DDA has exactly the same size as the standard DDA but with a position translation up to the object for positioning its own components.

VEMMI items are generally implicitly positioned with respect to the width or the height of the nearest item on the left or above.

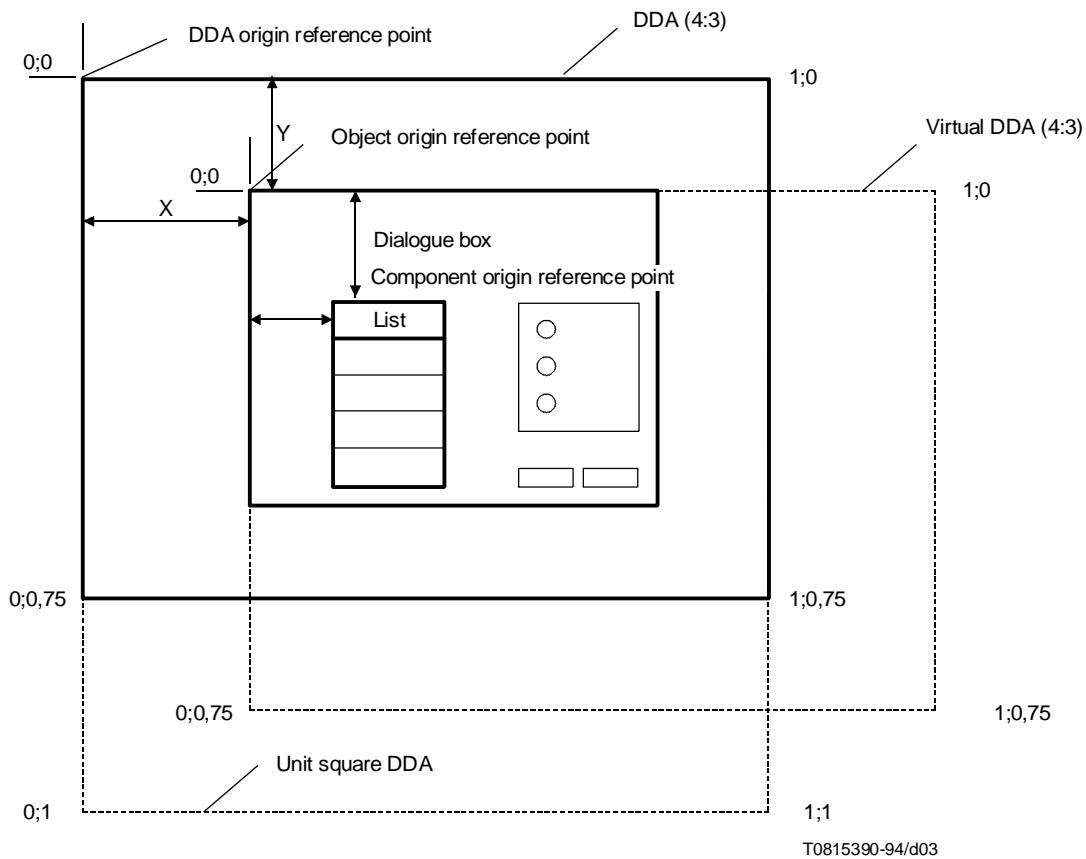


FIGURE 3/T.107
VEMMI positioning NDC space

4.6.2 Dimensioning

The standard plane shall continue to support the coordinate system used for dimensioning at the underlying platform.

The VEMMI objects plane and VEMMI objects and components shall support a dimensioning system based NDC.

The width and the height of a VEMMI object or of a VEMMI component, expressed in NDC, are referred with respect to their positioning reference point (upper left-hand corner), see Figure 4.

In a VEMMI element, data content follows the rules for the coding of its relevant data syntax including a possible picture placement or positioning using the relevant coordinate system.

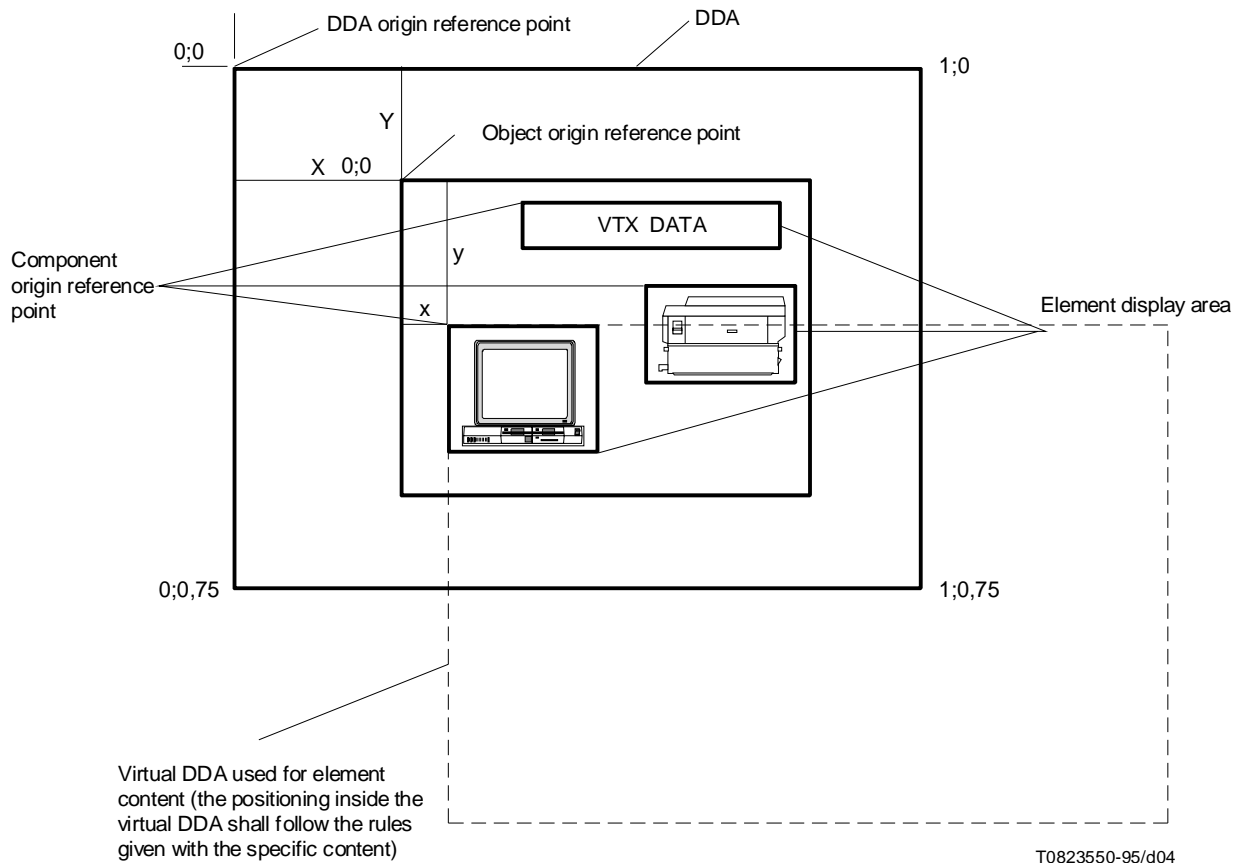


FIGURE 4/T.107
VEMMI positioning system

4.7 VEMMI elements states and state parameters

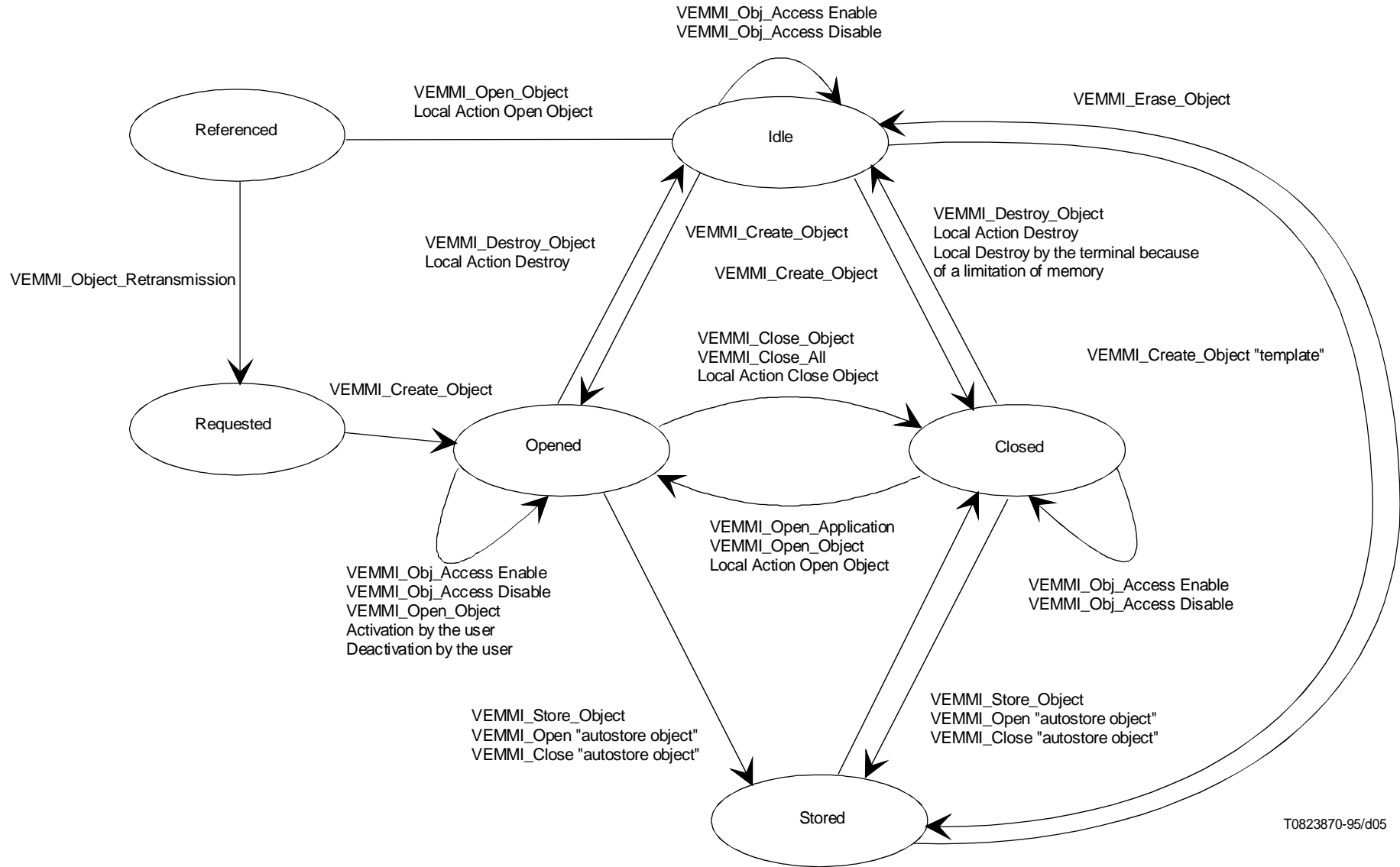
4.7.1 Object

An object can adopt different logical states with different state parameters. These states and state parameters may be the result of:

- user inputs;
- local actions;
- actions taken by the VEMMI local manager;
- commands from the VEMMI application directly or via a metacode object.

4.7.1.1 Definition of object states

Figure 5 shows the commands which can be used by the VEMMI application to change the object state or its state parameters.



T0823870-95/d05

FIGURE 5/T.107
State diagram for objects

In general, the following logical states for objects are defined with certain restrictions applying to specific objects (see clause 7).

– *Opened*

An object which exists in the terminal is displayed.

Opening an object is initiated by the VEMMI application, either as initial state at the time of the object creation, by a direct VEMMI command any time during the application or by a local action which is associated to a component and triggered by user interaction.

– *Closed*

An object which exists in the terminal is not displayed.

Closing an object is initiated by the VEMMI application, either as initial state at the time of the object creation, by a command any time during the application or by a local action which is associated to a component and triggered by user interaction.

– *Referenced*

An object which does not exist on the terminal was referenced by a VEMMI_Open_Object command or the local action “Open object”. The terminal shall request the retransmission of the specified object with the VEMMI_Object_Retransmission command.

– *Requested*

After the attempt to open a non-existing object, a request for object retransmission shall be sent from the terminal to the VEMMI application. While the terminal is waiting for the creation of the requested object, the object is in the requested state.

– *Stored*

A copy of an object (including all its components), which was used in an earlier session or in the actual opened session and has been stored permanently in the terminal (e.g. on hard disk).

A stored object shall implicitly have the state closed. Stored objects are made available for object operations with the command Open Application. Then they behave like received closed objects.

If an object that was previously stored in the terminal is requested to be stored again by the host application, the newer one shall replace the older one with all its parameters and states.

Stored objects can be deleted with the command VEMMI_Erase_Objects.

An object which is in the closed state may be destroyed from terminal memory by a local decision of the VEMMI local manager (e.g. because of a limitation of memory). The VEMMI application may be informed about this local destroy using the VEMMI error command “Object Destroy Indication”. If the terminal receives a VEMMI_Open_Object command referring to an object that has been destroyed or if a local action “Open object” referring to a destroyed object is executed, the terminal shall request the retransmission of the object from the application, using the VEMMI_Object_Retransmission command. The VEMMI application shall then create the requested object again using the VEMMI_Create_Object command applying its current state and its current state parameters within the VEMMI application. A VEMMI_Create_Object command resulting from a VEMMI_Object_Retransmission request shall always create the object in the open state.

4.7.1.2 Definition of object state parameters

In general, the following state parameters for objects are defined with certain restrictions applying to specific objects (see clause 7).

– *Active*

An object currently has the input focus. The user inputs always refer to the active object. The active object is on top of the DDA. If the active object is inaccessible it is not possible for the user to interact with it.

The active state management is described in 4.10.1.

– *Inactive*

The opposite of active.

- *Accessible*

The user can interact with the object.

Enabling the interaction with an object is initiated by the VEMMI application, either as initial state at the time of the object creation, by a VEMMI command any time during the application or by a local action which is associated to a component and triggered by user interaction.

- *Inaccessible*

The opposite of accessible.

The active/inactive state parameter applies only to components which are in the opened state. The accessible/inaccessible state parameter is applicable to open and closed components. Any combination of the state parameters active/inactive and accessible/inaccessible can exist for an open component.

4.7.2 Component

4.7.2.1 Definition of component states

A component can adopt different logical states with different state parameters. These states and state parameters may be the result of:

- user inputs;
- local actions;
- commands from the VEMMI application.

Figure 6 shows the possible states for components. The states and state parameters of components can be changed during the applications using the VEMMI_Modify_Component command.

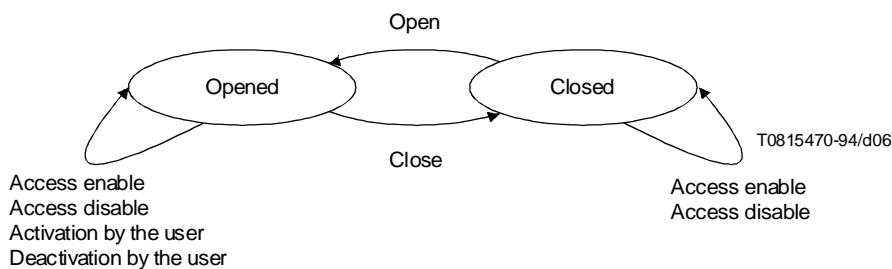


FIGURE 6/T.107
State diagram for components

In general, the following logical states for components are defined with certain restriction applying to specific components (see clause 7).

- *Opened*

A component which exists within an object is displayed.

Opening a component is initiated by the VEMMI application, either as initial state at the time of the object creation, by a VEMMI_Modify_Component command any time during the application or by a local action which is associated to a component and triggered by user interaction.

- *Closed*

A component which exists in the terminal is not displayed.

Closing a component is initiated by the VEMMI application, either as initial state at the time of the object creation, by a VEMMI_Modify_Component command any time during the application or by a local action which is associated to a component and triggered by user interaction.

4.7.2.2 Definition of component state parameters

In general, the following state parameters for components are defined with certain restrictions applying to specific components (see Clause 7).

– *Active*

A component has currently the input focus. The user input always refers to the active component within the active object. If the active component is inaccessible, it is not possible for the user to interact with it.

The active state management is described in 4.10.1.

– *Inactive*

The opposite of active.

– *Accessible*

The user can interact with the component if it is part of the current active object.

Enabling the user interaction with a component is initiated by the VEMMI application, either as initial state at the time of the object creation, by a VEMMI_Modify_Component command any time during the application or by a local action which is associated to a component and triggered by user interaction.

– *Inaccessible*

The opposite of accessible.

4.8 Local action management

VEMMI commands are generally issued by the VEMMI application to change the state or the state parameters of an object or component. Only a very limited number of changes or object or component states or their parameters can be initiated by the terminal.

To enable the VEMMI application to interact with the user, the terminal shall provide the capability to report the user inputs on the access or validation of components.

The result of these interactions is that changes of object or component states, which are the result of user inputs, could be delayed because the user inputs were sent to the VEMMI application first and then the application could react with a VEMMI command to change the state. This procedure might cause unacceptable response times on data links with lower data throughput. To improve the response times, a set of local actions has been defined.

Local actions are part of the component definitions and stored in the terminal at the time of the object creation. Each local action can consist of a list of “Element specific commands”, a list of “Report commands” and a list of “General commands” in any order. The following commands are defined:

a) *Element specific commands*

- open components of the parent object (parameter: list of CINs);
- close components of the parent object (parameter: list of CINs);
- open objects (parameter: list of OINs);
- close objects (parameter: list of OINs);
- change components state parameter of a component in the parent object to inaccessible (parameter: list of CINs);
- change components state parameter of a component in the parent object to accessible (parameter: list of CINs);
- destroy objects (parameter: list of OINs);
- open blocking object (parameter: OIN).

b) *Report commands*

- report OIN, CIN;
- report the current values of the component;

- report the values of all components in the parent object;
- report all values of all components in the parent objects that have changed (also via a VEMMI_Modify_component) since the last report or since the object creation if this is the first report.

c) *General commands*

- user lock;
- restore initial values of all input components in the parent object. This command should only be applied if the object was created with the “StoreInitialValue” attribute set true.

These commands are part of the object definition and they can appear in the local action of a component in any possible order. The order of appearance shall be equal to the order of their execution. There are two specific trigger events which induce the performance of local actions:

- activation of a component;
- validation of a component.

A list of “Element specific commands”, a list of “Report commands” and a list of “General commands” can be used to specify one local action for a component. No command of a specific type is mandatory for a local action.

There are two specific trigger events which induce the performance of local actions:

- activation of a component;
- validation of a component.

A local action can be associated to each of these trigger events.

4.9 Memory considerations

VEMMI terminals are supposed to have memory capabilities correctly dimensioned to provide a satisfactory VEMMI service in close relationship with their data type contents handling.

When there is not enough memory to execute a VEMMI command or a local action, the local manager may decide to destroy some closed objects. This mechanism is terminal dependent.

The VEMMI terminal shall send a VEMMI_Error (Out of memory) message to the VEMMI application when it does not have the possibility to release enough memory to process a received object description with a VEMMI_Create_Object command, VEMMI_Modify_Component command or VEMMI_Open_Object command. In that case it may ignore further VEMMI_Create_Object commands as long as the VEMMI application has not released a significant part of memory using VEMMI_Close_Object or VEMMI_Destroy_Object commands. The VEMMI application may also decide to disconnect the line if the application cannot be performed correctly due to a lack of memory in the terminal; however, the user shall always be informed of that decision by means of a message in a Message Box.

4.10 Common rules for object handling

4.10.1 Active state and focus management

A VEMMI object can become active as the result of:

- a) opening by the VEMMI application;
- b) user access;
- c) a local action with the element specific command “open object”;
- d) closing the active object. The open object that was most recently active becomes the new active object.

The active state shall be given to the last opened VEMMI object up to a user access to a different opened VEMMI object. When an accessible opened VEMMI object is accessed by the user, the active state shall be given to this VEMMI object by the local manager. The active state is then handled by the local manager up to a new VEMMI object opening or re-opening action from the VEMMI application. There should always be one single active object on the DDA. The active object can be inaccessible.

If objects use a common area on the DDA, the successive object activation leads to partial or full object overlapping. The active object shall always be fully visible to the user just after its activation. The previous active object is logically just beneath the active object. This rule shall apply to all previously active objects.

When set active, the VEMMI object and/or one of its components shall receive its focus from the local manager. The active state of the VEMMI object and/or of the VEMMI components shall be clearly visually indicated to the user. Within the active VEMMI object, not more than one accessible VEMMI component shall receive the active state and the focus.

The first time an object is displayed and consequently activated, the component which gets the focus may be specified by the VEMMI application within the object description. If there is no component specified or the component specified is not opened, then the terminal shall give the focus to the first created opened and accessible component. The other times the manner of giving the focus on activation of an object to one of its components is terminal dependent (same rule as above, memorization of the last accessed component, the nearest component of the user access to the parent object).

4.10.2 Behaviour of the modal mode

For a modal active object any user interaction with another VEMMI object of the same VEMMI application is forbidden by the terminal, any tentative attempt is indicated to the user (buzzer).

All the commands received from the host and all the local actions resulting from user interactions with the components of the modal object are executed.

4.10.3 Size considerations and clipping

The data content of a VEMMI element shall be sent with dimensions and a possible position compatible with the size description of its container (object or component). A VEMMI terminal shall clip non-consistent data content to the overall dimensions of the container.

For bitmap data content specific rules apply that are given with the description of the elements.

4.11 Local object storage

A host application has the possibility to request the load of objects locally stored in the terminal. Such a set of objects is application specific and contains typically those parts which remain unchanged for a longer time period and it is therefore more efficient not to transmit them in each dialogue session to the terminal. The loaded objects are processed in the same way as if they were received during the present session. Upon a host request they can be restored again in order to keep them up to date. The objects storage does not imply a removal of the objects from the terminal application, only a copy of the objects is stored. The dialogue may continue and is not affected by the objects storage. An object set is identified with the application identifier.

The timestamp is received from the host at the beginning of the application. It is stored with the objects and reused in later sessions. The timestamp value is never calculated by the terminal, it is only used in comparison to find outdated objects. Whenever an object is stored locally in the terminal it shall have a timestamp associated. The value of this timestamp shall be determined as follows:

- if the object was created by the host (by VEMMI_Create_Object) during the current session or if any component of the object was modified (by VEMMI_Modify_Component) during the current session, the object shall be stored with the timestamp value given by the VEMMI_Open_Application command for the current application;
- in all other cases the objects timestamp shall remain unchanged. (The object was loaded without change from the local storage during this session.)

Two attributes which are part of the object definition can be used additionally to request the object storage either immediately after the object creation (template) or at the end of the application (autostore).

A stored set of objects contains the following:

- the application identifier;
- the objects including their components and the associated timestamp;

- the referenced files (if a referenced file is not available in the terminal, the object is considered incomplete and shall not be stored);
- the relevant loaded colour table.

The storage format is terminal specific because the objects are loaded and stored only by the same terminal.

The terminal implementation should provide the user with the appropriate means to manage the local storage and typically the user may limit its maximum size.

4.12 Symbolic directory names

In order to support the host application to locate the correct files, symbolic directory names are defined. They can be used as part of the “filename” parameter. The terminal maps them on real directory respective filenames. In the following example, a host application uses a symbolic directory name to identify a bitmap file:

VEMMI_Create_Object (OIN = 7, ..., filename = “\$CD\ARTISTS\DANCER\LISA.BMP”, pict file type = BMP).

The string “\$CD” identifies the CD-ROM drive. The terminal substitutes this symbolic name with the real drive identification. The complete file identification might be: “H:\ARTISTS\DANCER\LISA.BMP”.

The following symbolic names are defined, the application may use other symbolic names:

Name	Meaning
\$APL	Parent directory of the applications stored in the terminal
\$FTD	Parent directory of downloaded files
\$CD	Drive directory name of a CD-ROM drive

These symbolic names can be used in the parameter “filename” of the VEMMI commands. The symbolic name shall appear only once in the filename parameter. For the separation of the symbolic name from additional subdirectories and from the filename, the character “\”, code 5/12, should be used.

4.13 Specific rules for dedicated terminals

All service primitives including their parameters shall be understood also by a dedicated terminal. Regarding the execution of the commands related to specific functionality, a dedicated terminal may apply graceful degradation mechanisms.

The following functionality can be gracefully degraded by a dedicated terminal:

- colour definition;
- in-text attributes (fonts, colours, font attributes);
- local storage;
- NDC-positioning and dimensioning.

For dedicated VEMMI terminals only able to handle Videotex data contents on a character basis (alphamosaic, DRCS, Photographic Profile 1), VEMMI objects and components can be physically positioned and dimensioned to the nearest corresponding display character position. Consequently, to avoid side effects between VEMMI objects, NDC object positions should be expressed in a direct multiple of character positions for VEMMI application using only this type of Videotex data contents.

For dedicated VEMMI terminals only able to handle Videotex data contents on a character basis, VEMMI can be used with 80-character mode making use of the appropriate SWITCHING SEQUENCE or the DEFINE FORMAT Videotex Presentation Data Element (VPDE) to switch to the desired Data Syntax. Eighty-character mode is not mandatory for a VEMMI terminal.

Dedicated VEMMI terminals shall recognize, but not necessarily process the whole or part of the VEMMI high quality text attributes they are not able to display. These terminals may then ignore the text attributes and display text using the system font, in the available system colour, on the system background.

Dedicated VEMMI terminals shall support the different kinds of VEMMI objects, with possible restrictions applying to the operative objects. As a file management system is not mandatory, they can manage the files they receive in the VEMMI_Resource_Transfer command in their own internal representation.

Dedicated terminal should support the minimum kernel datatypes defined in Annex C. However, when used in a retrieval service where the standard mode is the Videotex mode and when the terminal does not support the Videotex photographic syntax (to be enquired using the TFI) the support of JPEG and GIF is not mandatory. When the terminal supports the Videotex photographic syntax the support of JPEG is mandatory; the support of GIF is recommended. In the same service conditions, when a terminal does not support the Videotex audio syntax the support of WAVE and MIDI is not mandatory. When the terminal supports the Videotex audio syntax the support of WAVE is mandatory; the support of MIDI is recommended.

Dedicated terminals need not support the stretching of bitmap data to fill the element display area. If the presentation attribute for a bitmap is set to "stretched", the terminal may present the bitmap tiled.

5 Service description

This clause describes the service characteristics offered by the VEMMI.

The service is defined between a terminal and a remote application. No assumption is made about the way the terminal accesses to the service (Public Switched Telephone Network (PSTN)/Packet Switched Public Data Network (PSPDN)/Integrated Services Digital Network (ISDN)) and the way the connection is established.

All VEMMI service elements or VEMMI commands shall only be issued on an established network connection. Before sending the first VEMMI command, the VEMMI application should issue a Terminal Facility Identifier (TFI) request to ensure that the terminal is able to support the VEMMI standard.

All VEMMI service elements are mandatory for a VEMMI terminal. A VEMMI terminal shall understand all VEMMI service elements including all their parameters.

The VEMMI services are divided into two groups:

- services initiated by the VEMMI application;
- services initiated by the VEMMI terminal.

A second source of commands for a VEMMI terminal is the metacode object. See Table 2.

The following services are confirmed explicitly:

- VEMMI_Identify_Term_Cap is confirmed with VEMMI_Identify_Term_Cap_Resp;
- VEMMI_Open_Application is confirmed with VEMMI_Open_Application_Resp;
- VEMMI_Store_Objects is confirmed with VEMMI_Store_Objects_Resp.

There is no explicit confirmation in any other of the service elements.

All VEMMI services, except VEMMI_Open_Object, which refer to a non-existing object shall be ignored by the VEMMI terminal. If a VEMMI_Open_Object refers to a non-existing object, the terminal shall request its retransmission.

NOTE – In the following subclause, all VEMMI services are described but only those parameters are listed which are necessary to understand the service. For a complete parameter list, see clause 8.

TABLE 2/T.107

VEMMI services

VEMMI Services	Initiated Application/Terminal	Function
VEMMI_Set_Translation_Mode	Application/Terminal	Sets the translation mode for VEMMI data
VEMMI_Open	Application	Switch to VEMMI mode and reset
VEMMI_Close	Application	Switch to standard mode and reset
VEMMI_Suspend	Application	Switch to standard mode
VEMMI_Resume	Application	Switch to VEMMI mode
VEMMI_Identify_Term_Cap	Application	Request for terminal capabilities
VEMMI_Identify_Term_Cap_Resp	Terminal	Response to the "identify terminal capability" request
VEMMI_Set_Options	Application	Set options in the terminal
VEMMI_Create_Object	Application	Object definition
VEMMI_Open_Object	Application	Display object
VEMMI_Open_Blocking_Object	Application	Open object
VEMMI_Close_Object	Application	Clear an object from the screen
VEMMI_Close_All	Application	Clear all the objects from the screen
VEMMI_Destroy_Object	Application	Clear object in the terminal memory
VEMMI_Obj_Access_Disable	Application	User access is not permitted
VEMMI_Obj_Access_Enable	Application	User access is permitted
VEMMI_Modify_Component	Application	Modify components characteristics
VEMMI_Obj_Location_Change	Application	Defines a new object position
VEMMI_Load_Col_Table	Application	Loads a colour table
VEMMI_Reset_Col_Table	Application	Resets a colour table
VEMMI_Open_Application	Application	Opens an application
VEMMI_Open_Application_Resp	Terminal	Response to "open application" request
VEMMI_Delete_Outdated_Objects	Application	Deletes outdated objects
VEMMI_Store_Objects	Application	Stores a set of objects
VEMMI_Store_Objects_Resp	Terminal	Response to a "store object" request
VEMMI_Erase_Objects	Application	Delete objects from the local storage
VEMMI_User_Lock	Application	User inputs are not permitted
VEMMI_User_Unlock	Application	User inputs are permitted
VEMMI_Object_Retransmission	Terminal	Request for object retransmission
VEMMI_User_Data	Terminal	User data in VEMMI mode
VEMMI_Error	Terminal	Error condition or situation
VEMMI_Resource_Transfer	Application	Resource transfer
VEMMI_Resource_Transfer_Abort	Terminal	Resource transfer abort

5.1 Service elements initiated by the VEMMI application and the terminal

If issued by the VEMMI application, this service element shall be used to indicate that the VEMMI data sent from the application to the terminal are transcoded using a specified translation mode.

If issued by the VEMMI terminal, this service element shall be used to indicate that the VEMMI data sent from the terminal to the application are transcoded using a specified translation mode.

Since a VEMMI terminal usually begins operation, by default, in either a 7-or 8-bit environment it shall not be mandatory for a VEMMI application to be sent as a first VEMMI command in a VEMMI session a VEMMI_Set_Translation_Mode to switch the terminal to a desired translation mode. If the application cannot be sure that all terminals that are connecting operate in a specific environment, it should send a VEMMI_Set_Translation_Mode to ensure that the terminal will interpret the VEMMI commands correctly. See Table 3.

Typically the VEMMI_Set_Translation_Mode is the first VEMMI command of a VEMMI session but it can be used at any time during a VEMMI session.

NOTE – The translation mode can be different in both directions.

TABLE 3/T.107

Parameters

Parameters	Description
TranslationMode	One byte specifying the desired translation mode

The TranslationMode parameter can have the following values:

- 2/0: no translation (transparent);
- 2/1: 3 in 4 encoding (see Recommendation T.101 [4]);
- 2/2: 7-shift encoding (see Recommendation T.101 [4]).

5.2 Service elements initiated by the VEMMI application

5.2.1 VEMMI_Open

This service element shall be used to switch the VEMMI terminal to the VEMMI operation mode. The switching mechanism is defined in 4.4.3. In VEMMI operation mode, the mechanism described in 4.4.2 shall apply.

In addition to the switch to VEMMI operation mode, the service element shall perform the following resets:

- close an open application;
- destroy all objects from the set of objects that are in the “opened” or “closed” state;
- all opened objects are cleared from the display;
- release all identification numbers for resource objects;
- set the colour table to the predefined values.

Objects stored in a local storage are not affected.

If this command is received in the VEMMI mode, no switch is necessary but only the resets are performed. See Table 4.

TABLE 4/T.107

Parameters

Parameters	Description
Version	Identifier of the VEMMI version
PrivateMode	Private parameter which can be used in some “page-based” Videotex services. This mode is not described in this Recommendation

5.2.2 VEMMI_Close

This service element shall be used to switch the VEMMI terminal to the standard mode. The switching mechanism is defined in 4.4.3. In standard operation mode, the mechanism described in 4.4.1 shall apply.

In addition to the switch to standard mode, the service element shall perform the resets defined in 5.2.1. See Table 5.

TABLE 5/T.107

Parameters

Parameters	Description
None	

5.2.3 VEMMI_Resume

This service element shall be used to switch the VEMMI terminal to the VEMMI operation mode. The parameters of this service element are the same as for the VEMMI_Open (see 5.2.1). With this service element no reset shall be performed. See Table 6.

TABLE 6/T.107

Parameters

Parameters	Description
Version	Identifier of the VEMMI version
PrivateMode	Private parameter which can be used in some “page-based” Videotex services. This mode is not described in this Recommendation

5.2.4 VEMMI_Suspend

This service element shall be used to switch the VEMMI terminal to the standard mode. No reset shall be performed. See Table 7.

TABLE 7/T.107

Parameters

Parameters	Description
None	

5.2.5 VEMMI_Identify_Term_Cap

This service element shall be used to request identification of the capabilities of the terminal. See Table 8.

The following information can be obtained:

- VEMMI version identification;
- local storage support (supported/not supported);
- list of preferred user languages (string);
- system type (e.g. IBM compatible, Macintosh, ...), operating system (e.g. MS-DOS, OS2, Microsoft Windows, ...), operating system version (e.g. 3.1, ...) (string);
- a list of supported datatypes (reference to the VEMMI content encoding identification catalogue).

If the DesiredInformation parameter is not present, the terminal shall respond giving a complete list that identifies all its capabilities.

TABLE 8/T.107

Parameters

Parameters	Description
DesiredInformation	The type of information that is requested from the terminal

5.2.6 VEMMI_Set_Options

This service element shall be used to set options in the terminal. See Table 9.

The following options can be set:

- if the terminal supports different kinds of text encoding, the desired encoding standard can be selected.

TABLE 9/T.107

Parameters

Parameters	Description
TextStandard	Reference to a text encoding standard in the VEMMI content encoding identification catalogue which shall be used at the beginning of the encoding of datatypes string and character

5.2.7 VEMMI_Create_Object

This service element shall be used to define a VEMMI object and its components in the VEMMI terminal. The VEMMI terminal shall create the object and shall apply the object state and state parameters given in its definition.

OINs shall be unique at one time in an application. The creation of an object with an already existing OIN shall cause the destruction of the previously defined object.

CINs shall be unique within an object. The definition of a component with an already existing CIN shall cause the destruction of the previously defined component. See Table 10.

TABLE 10/T.107
Parameters

Parameters	Description
Object Identification Number	
Template	If an application is open the object shall be stored immediately after creation
Autostore	If an application is open the object shall be stored at the reception of a VEMMI_Open, a VEMMI_Close or a VEMMI_Open_Application

5.2.8 VEMMI_Open_Object

This service element shall be used to set an object to the opened state in the VEMMI terminal. The terminal shall display the specified object immediately. If the object is not present in the terminal, it shall request the object by sending a VEMMI_Object_Retransmission request to the VEMMI application with the specified OIN, in this case “Rule A” as described in 5.3.2. shall be used. The application shall then create the object again, applying its current state and state parameters.

A VEMMI_Open_Object command referring to an already opened object causes the activation of the object.

If a VEMMI_Open_Object is applied to a resource object, the terminal shall verify whether the object is available. If not, it shall be requested. No further action shall be performed.

If a VEMMI_Open_Object is applied to a metacode object, the terminal executes the object content. If not present it shall be requested.

If a VEMMI_Open_Object is applied to an operative object, the terminal executes the program. If not present it shall be requested. See Table 11.

TABLE 11/T.107
Parameters

Parameters	Description
Object Identification Number	

5.2.9 VEMMI_Open_Blocking_Object

This service element performs the same as VEMMI_Open_Object except that the retransmission case “Rule B” as described in 5.3.2. shall be used.

5.2.10 VEMMI_Close_Object

This service element shall be used to close a VEMMI object in the VEMMI terminal. The VEMMI terminal shall clear the indicated object from the screen but keep it in its memory. The current state parameters and values of the components shall not be changed.

When receiving a VEMMI_Close_Object indication referring to an active object, the new active object in the VEMMI terminal shall be the opened object that was most recently active in the terminal.

For objects that are not displayable or are already in a closed state, the VEMMI_Close_Object has no effect. See Table 12.

TABLE 12/T.107

Parameters

Parameters	Description
Object Identification Number	

5.2.11 VEMMI_Close_All

This service element shall be used to close all the VEMMI objects in the VEMMI terminal. A VEMMI_Close_All service element shall not change the current operation mode of the terminal. See Table 13.

TABLE 13/T.107

Parameters

Parameters	Description
None	

5.2.12 VEMMI_Destroy_Object

This service element shall be used to remove a VEMMI object from a set of objects that are in the state “opened” or “closed”. The terminal shall remove all data belonging to this object from its memory. If the VEMMI_Destroy_Object is applied on an opened object, it shall be closed by the local manager and then destroyed. If the object was active, the new active object in the VEMMI terminal shall be the opened object that was most recently active in the terminal. See Table 14.

TABLE 14/T.107

Parameters

Parameters	Description
Object Identification Number	

5.2.13 VEMMI_Obj_Access_Disable

This service element shall be used to restrict the user access to an object in the VEMMI terminal. The access shall be restricted until a VEMMI_Obj_Access_Enable service element referring to the same object is received by the VEMMI terminal.

If the object was active, the new active object shall be the open object that was most recently active in the terminal. A VEMMI_Obj_Access_Disable Indication referring to an active object shall interrupt the user interaction with this object. See Table 15.

TABLE 15/T.107

Parameters

Parameters	Description
Object Identification Number	

5.2.14 VEMMI_Obj_Access_Enable

This service element shall be used to permit user access to an object in the VEMMI terminal. The access shall be permitted until a VEMMI_Obj_Access_Disable service element referring to the same object is received by the VEMMI terminal.

A VEMMI_Obj_Access_Enable Indication received in the VEMMI terminal and referring to an inactive object shall not change this state parameter. See Table 16.

TABLE 16/T.107

Parameters

Parameters	Description
Object Identification Number	

5.2.15 VEMMI_Modify_Component

This service element shall be used to modify one or several VEMMI components belonging to a VEMMI object created into the VEMMI terminal. The possible modifications in VEMMI components are defined in the component definition tables of 9.6.

If a VEMMI_Modify_Component command is applied to an opened object, it shall have an immediate visual effect in the VEMMI terminal if the modified part is visible.

A VEMMI_Modify_Component command shall not change the active/inactive state parameter of the object in the VEMMI terminal. See Table 17.

TABLE 17/T.107

Parameters

Parameters	Description
Object Identification Number	
Modification description	

5.2.16 VEMMI_Obj_Location_Change

This service element shall be used only to change the position of a VEMMI object (not component) in the VEMMI, terminal screen. If a VEMMI_Obj_Location_Change command is referring to a VEMMI closed object no visible effects shall be encountered. This service element shall not change the active/inactive state parameter of an object.

A VEMMI_Obj_Location_Change service element shall never be applied to an Application Bar. See Table 18.

TABLE 18/T.107

Parameters

Parameters	Description
Object Identification Number	
XPos	New horizontal position
YPos	New vertical position

5.2.17 VEMMI_Load_Col_Table

This service element shall be used by the host to define colours in the colour table.

The colours are defined consecutively, started with the index "ColEntry". The terminal shall send an error message to the host if the requested colours cannot be defined, e.g. because the device supports less colours than the host requested. See Table 19.

TABLE 19/T.107

Parameters

Parameters	Description
ColEntry	Colour index from which the colours are defined in ascending order
ColRGBList	The list consists of triplets which give the red, green and blue component of a colour. Each component value is in the range 0 .. 63. The value 0 means no intensity for the component, the value 63 means maximum intensity

NOTE – If this service is not supported by the terminal, this shall be reported to the using application using the VEMMI_Error "Service not supported".

5.2.18 VEMMI_Reset_Col_Table

This service element shall be used by the host to reset all previously defined colours. This command has no parameters. Only the default colours (indices 0 .. 15) remain defined. If the default colours have been redefined with VEMMI_Load_Col_Table, this command sets the default colours to their predefined values. See Table 20.

TABLE 20/T.107

Parameters

Parameters	Description
None	

The default colours are given in Table 21.

TABLE 21/T.107

Parameters

Colour index	Red	Green	Blue	Colour
0	0	0	0	Black
1	0	0	42	Blue
2	0	42	0	Green
3	0	42	42	Cyan
4	42	0	0	Red
5	42	0	42	Magenta
6	42	21	0	Brown
7	42	42	42	White
8	21	21	21	Grey
9	21	21	63	Light blue
10	21	63	21	Light green
11	21	63	63	Light cyan
12	63	21	21	Light red
13	63	21	63	Light magenta
14	63	63	21	Yellow
15	63	63	63	Bright white

NOTE – If this service is not supported by the terminal, this shall be reported to the using application using the VEMMI_Error “Service not supported”.

5.2.19 VEMMI_Open_Application

This service element shall be used by the host to load a set of objects and to define the name and attributes of the current application. Additional support can be provided using symbolic directory names as described in 4.12. Locally stored object sets are identified by the application identifier. A timestamp is used to associate version information with the objects of this application. See Table 22.

TABLE 22/T.107

Parameters

Parameters	Description
AppId	String with the application name
ApplAddData	Additional application identification data
Timestamp	Time in seconds since 1.01.1970, Greenwich Mean Time (GMT)

The parameter AppId shall contain the application name. Additional information, like name of the company or the organization to which the application belongs or the author of the application can be given with the parameter ApplAddData, which is not mandatory. The terminal does not need to analyze the content of AppId, it is only used to identify the requested application.

Application providers should take care to choose unique application identifiers, in order to avoid overwriting of other applications.

5.2.20 VEMMI_Delete_Outdated_Objects

This service element shall be used by the host to request the deletion of outdated local objects of the current application from the terminal storage. See Table 23.

If the host issues a VEMMI_Delete_Outdated_Objects with the timestamp requirements for specific objects, the terminal shall react as follows:

- if an object is referenced with a timestamp value higher than the timestamp value of the locally stored object, this object shall be deleted from the local storage. The host application can thus be certain that outdated objects are no longer used.

TABLE 23/T.107

Parameters

Parameters	Description
UpdateList	This parameter contains groups, each of two elements, a timestamp and a set of objects

In order to update an application partially, this command is typically sent following a VEMMI_Open_Application.

5.2.21 VEMMI_Store_Objects

This service element shall be used by the host to request the terminal to store the current loaded objects or a part of them. The timestamp of the current open application, supplied with the VEMMI_Open_Application command, is associated to each object. See Table 24.

TABLE 24/T.107

Parameters

Parameters	Description
Object Identification Number	

If the parameter is absent, all currently loaded objects are stored.

5.2.22 VEMMI_Erase_Objects

This service element shall be used by the host to request the deletion of objects of the current application from the local storage of a terminal. To increase security implementation, the VEMMI terminal could ask the user before executing the deletion. (e.g. one time at the beginning of the application or with every deletion). If no application is open, the command has no effect. See Table 25.

TABLE 25/T.107

Parameters

Parameters	Description
Object Identification Number	

5.2.23 VEMMI_User_Lock

This service element shall be used to restrict any user input in the terminal until a VEMMI_User_Unlock is sent. See Table 26.

TABLE 26/T.107

Parameters

Parameters	Description
None	

5.2.24 VEMMI_User_Unlock

This service element shall be used to permit user inputs in the terminal. See Table 27.

TABLE 27/T.107

Parameters

Parameters	Description
None	

5.2.25 VEMMI_Resource_Transfer

This service element shall be used to transfer files referenced by VEMMI resource objects from a VEMMI application to a VEMMI terminal. See Table 28.

A file can be downloaded using one or more VEMMI_Resource_Transfer commands. A parameter of the VEMMI_Resource_Transfer command (TransferID) associates the command to a given resource transfer. This allows several resource transfers to be performed simultaneously with commands pertaining to different resource transfers interleaving each other without interference.

The VEMMI_Resource_Transfer commands pertaining to the same resource transfer are numbered consecutively using the parameter BlockNumber.

The first block of a given resource transfer shall always contain the following header information of the file:

- filename;
- filesize;
- number of VEMMI_Resource_Transfer commands used to transmit the file;
- date of creation.

The TransferType parameter of the VEMMI_Resource_Transfer command indicates if the VEMMI_Resource_Transfer command of a given resource transfer is:

- the first command;
- an intermediate command; or
- if the resource transfer is aborted by the VEMMI application.

TABLE 28/T.107

Parameters

Parameters	Description
TransferType	First command, intermediate command, abort
TransferID	Transfer identifier
BlockNumber	Number of the command within a given resource transfer
FileInformation	Header information about the file
TotalBlocks	Number of VEMMI_Resource_Transfer commands used to transmit a given file

5.3 Service elements initiated by the terminal

5.3.1 VEMMI_Identify_Term_Cap_Resp

This service element shall be used to respond to a VEMMI_Identify_Term_Cap request. See Table 29.

The following information can be given:

- VEMMI version identification;
- local storage support (supported/not supported);
- list of preferred user languages (string);
- system type (e.g. IBM compatible, Macintosh, ...), operating system (e.g. MS-DOS, OS2, Microsoft Windows, ...), operating system version (e.g. 3.1, ...) (string);
- a list of supported datatypes (reference to the VEMMI content encoding identification catalogue).

TABLE 29/T.107

Parameters

Parameters	Description
SupportedVEMMIVersions	This parameter contains the VEMMI version identification
ContentList	The list gives references of the supported datatypes in the VEMMI content encoding identification catalogue
LocalStorage	Identifies the support of the local storage facility
PreferredUserLanguages	Languages preferred by the user (for multilingual applications) The first language in the list is the most preferred. The list may contain only one language
SystemInfo	Information to support the correct use of operative objects (e.g. System Type, Operating System, Operating System Version)

5.3.2 VEMMI_Object_Retransmission

This service element shall be used to request the VEMMI application to retransmit a VEMMI object. It shall be used only after:

- the reception of a VEMMI_Open_Object command;
- a VEMMI_Open_Blocking_Object command;
- a local action “Open object”;

referring to an object that does not exist on the terminal.

For the further processing of VEMMI commands until the requested object is received, the terminal shall apply one of the following rules.

Rule A

- inform the user about possible delay due to a retransmission (terminal dependent);
- send a VEMMI_Object_Retransmission request;
- execute all VEMMI commands and local actions which do not refer to the requested object;
- memorize all VEMMI commands and local actions which refer to the requested object.

After the recreation and the opening of the requested object, the terminal shall:

- resume the execution of the possibly suspended local action;
- resume the execution of the possibly memorized VEMMI commands received in the order of their reception.

Rule B

- lock the user;
- inform the user about possible delay due to a retransmission (terminal dependent);
- send a VEMMI_Object_Retransmission request;
- suspend and memorize any execution of VEMMI commands or local actions.

After the recreation and the opening of the requested object, the terminal shall:

- resume the execution of the possibly suspended local action;
- resume the execution of the possibly memorized VEMMI commands received in the order of their reception;
- unlock the user.

See Table 30.

TABLE 30/T.107

Parameters

Parameters	Description
Object Identification Number	

5.3.3 VEMMI_User_Data

This service element shall be used to send user data corresponding to one object to the VEMMI application. See Table 31.

TABLE 31/T.107

Parameters

Parameters	Description
Object Identification Number	
CompData	User inputs

5.3.4 VEMMI_Open_Application_Resp

This service element shall be used by the terminal to respond to a VEMMI_Open_Application command. See Table 32.

TABLE 32/T.107

Parameters

Parameters	Description
OpenAppResult	True: the application has been identified and the objects have been loaded False: the application has not been identified, no objects have been loaded. The parameters that were used with the VEMMI_Open_Application command are used to describe a new application environment

5.3.5 VEMMI_Store_Objects_Resp

This service element shall be used by the terminal to respond to a VEMMI_Store_Objects. See Table 33.

TABLE 33/T.107

Parameters

Parameters	Description
StoreResult	True: the objects have been stored False: the objects have not been stored

5.3.6 VEMMI_Error

This service element shall be used only by the VEMMI terminal to report different error situations to the VEMMI application.

A VEMMI_Error command with the parameter Type of Error = “Out of memory” is only referring to the specified object. Other objects created after this one may be stored in the terminal. See Table 34.

TABLE 34/T.107

Parameters

Parameters	Description
Object Identification Number	
ErrorType	
Component Identification Number	
ErrorComCode	Identification of the command that caused the error

The following types of errors are defined:

- General error:
A general error has occurred.
- Unknown VEMMI command:
The VEMMI command code does not exist.
- Erroneous VEMMI command:
The VEMMI command received does not have the mandatory parameters or they have erroneous values.
- Object syntax error:
The description of a VEMMI element is not correct in an object creation or component modification.
- Unexpected VEMMI command:
The VEMMI command received is correct but it occurs at the wrong point in time.
- Out of memory:
The terminal does not have enough memory to store the data corresponding to an object creation or component modification.
- Service not supported:
The service requested is not supported.
- Object not supported.
- Data content type not supported.
- Invalid colour index.
- File not found.
- Conversion to bitmap failed.
- Object destroy indication (a closed object was destroyed by the terminal).
- Out of local permanent storage space.

5.3.7 VEMMI_Resource_Transfer_Abort

This service element shall be used by the VEMMI terminal to abort a resource. See Table 35.

TABLE 35/T.107

Parameters

Parameters	Description
TransferID	Transfer identifier

6 VEMMI objects introduction

VEMMI objects offer a basic choice of dialogue elements needed by VEMMI applications.

The following VEMMI objects are defined in this Recommendation:

- application bar;
- button bar;
- pop-up menu;
- dialogue box;

- message box.
- operative object;
- bitmap resource object;
- videotex resource object;
- text resource object;
- font resource object;
- metacode object.

A VEMMI application can be designed using any of the defined objects. No particular VEMMI object or component is mandatory within a VEMMI application. All VEMMI objects may be multiple within a VEMMI application, except the Application Bar. Within an object, all components may be multiple except certain restrictions applying to specific components (see clause 7).

6.1 The application bar

The Application Bar allows the user to make a choice between the different VEMMI application parts and sub-application parts offered by the selected VEMMI application. When used, the Application Bar is unique and located either on the top (horizontal Bar) or the left side (vertical Bar) of the DDA.

6.1.1 Composition

The Application Bar is subdivided into three different logical groups of Menu Choice components. These groups differ in their behaviour and functionality. The three different groups are named:

- bar;
- pull-down menu;
- cascading menu.

The Bar is a horizontal or vertical list of Menu Choice components which represents the different parts of the VEMMI application.

The Pull-Down Menus are vertical lists of Menu Choice components which are associated to the same Menu Choice component of the Bar. The Pull-Down Menus represent the different sub-application parts of the VEMMI application.

The Cascading Menus are vertical lists of Menu Choice components which are associated to the same Menu Choice component of the Pull-Down Menu. The Cascading Menus represent the different sub-application parts of the VEMMI application.

6.2 The button bar

The Button Bar permits a choice among a set of alternatives, at a given time, during the execution of the VEMMI application. Each choice is represented by a Button. The Button Bar may be located anywhere in the DDA. The Button Bar can be horizontal or vertical.

6.2.1 Composition

The Button Bar is composed of a series of components, named Buttons.

6.3 The pop-up menu

The Pop-Up Menu offers appropriate choices and sub-choices for a given VEMMI element in its current context. The Pop-Up Menu may be located anywhere in the DDA.

6.3.1 Composition

The Pop-Up Menu is subdivided into two different logical groups of Menu Choice components. These groups differ in their behaviour and functionality. The two different groups are named:

- primary pop-up menu;
- cascading menu.

The Primary Pop-Up Menu is a vertical list of Menu Choice components which offers appropriate choices for a given VEMMI element in its current context.

The Cascading Menu is a vertical list of Menu Choice components associated to the same Menu Choice component of the Primary Pop-Up Menu. The Cascading Menu offers appropriate sub-choices for a given VEMMI element in its current context.

6.4 The dialogue box

The Dialogue Box is the object where the main interaction between the user and the VEMMI application takes place. To enable this interaction, a set of components is defined. These components can be classified as presentation or dialogue components.

Presentation components are inaccessible; their purpose is only to present the different dialogue components coherently and attractively.

Dialogue components permit the interaction between the user and the VEMMI application.

Five presentation components are defined:

- the Separator;
- the Frame;
- the Text Presentation Area;
- the Text component;
- the Graphic Output Area.

Nine dialogue components are defined:

- the Push Button;
- the Text Input Field;
- the Check Box;
- the Radio Button;
- the List Box;
- the Combination Box;
- the Slider;
- the Sensitive Area;
- the Sensitive Text.

6.4.1 Composition

6.4.1.1 The Separator component

A Separator is a horizontal or vertical solid line. Its goal is to visually separate different dialogue components within the Dialogue Box.

6.4.1.2 The Frame component

A Frame is a presentation element to visually separate a particular area of the Dialogue Box and its different components.

6.4.1.3 The Text Presentation Area component

The Text Presentation Area is a rectangular area in which text data is displayed. The goals of this component are:

- to present text information;
- to title or to label dialogue components;
- to present results from the VEMMI application execution.

6.4.1.4 The Text component

The purpose of this component is to split large text data in units (text components) and to define the necessary structural information (concatenation of the text components) in order to be displayed in a text area.

6.4.1.5 The Graphic Output Area component

The purpose of this component is to present graphical data to the user. Several graphical data encoding formats are supported.

6.4.1.6 The Sensitive Text component

The purpose of this component is to define the activation and validation operations for sensitive text strings.

6.4.1.7 The Push Button component

The purpose of the Push Button is to trigger a local action to be immediately performed.

6.4.1.8 The Text Input Field component

The purpose of the Text Input Field is to collect text data, entered by the user. It is a rectangular area composed of a text label associated to an input area.

6.4.1.9 The Check Box component

The purpose of the Check Box is to enter and to display an independent user choice. A Check Box keeps the value marked or unmarked independently of any other Check Boxes.

6.4.1.10 The Radio Button component

The purpose of the Radio Button is to enter and to display a user choice. The Radio Button permits a single choice among several possibilities offered in a Radio Button group. The marking of one Radio Button leads to the unmarking of the other Radio Buttons belonging to the same Radio Button group.

6.4.1.11 The List Box component

The purpose of the List Box is to offer a single or multiple choice among a list of text items. The list is, generally, not entirely visible to the user, so different controls are offered to scroll the list up and down.

6.4.1.12 The Combination Box component

The purpose of the Combination Box is to combine the functionality of a single choice List Box with the functionality of a Text Input Field. It contains a list of text items the user can scroll through to complete the Text Input Field. A parameter of the Combination Box specifies whether the Text Input Field content can be edited or not. If the Text Input Field content can be edited, the user can type text directly into the Text Input Field.

A variation of the Combination Box is a Drop Down Combination Box. It is composed of a Combination Box and a Push Button. Only the Text Input Field and the Push Button are displayed until the user selects the associated Push Button. The validation of the Push Button causes the display of the associated List Box.

6.4.1.13 The Slider component

The slider offers the selection of an analogue value by moving an adjustable marker on a slide bar between a minimum- and a maximum Value. The Intervals are set by the application.

6.4.1.14 The Sensitive Area component

The purpose of the Sensitive Area in the Dialogue Box is to offer a selection area associated to an Output Area.

6.5 Operative object

With this object an application references a program which will be linked to the VEMMI application. This object type provides a method to extend the capabilities of an application during runtime.

6.6 Bitmap resource object

A bitmap object contains either the bitmap definition itself or only a reference to a file with the bitmap definition.

6.7 Videotex resource object

A Videotex object contains either the Videotex content itself or only a reference to a file with the Videotex content.

6.8 Text resource object

This object defines text content as a resource which can be referenced via the “Text Identification Number” (TIN). It contains either the text content itself or a reference to a file with the text content.

6.9 Font resource object

This object combines a set of text attributes in a font resource which can be referenced via the FIN.

6.10 Metacode object

The metacode object contains VEMMI commands. This object provides an easy way to avoid unnecessary dialogue steps with the host application.

6.11 The message box

The purpose of the Message Box is to display information, not requested by a user but sent by the VEMMI application in response to an unexpected event, or when something undesirable might occur.

7 Functional description

7.1 General rules for the behaviour of elements

7.1.1 User interaction

The user shall have the possibility to access and activate the different VEMMI elements. The terminal shall also enable the user to change the values of the VEMMI elements and to validate these inputs.

7.1.2 Local actions and reports

The report of user inputs from the terminal to the server shall be induced by a report command defined in a local action by the VEMMI application. On executing a report command, the terminal shall send a VEMMI_User_Data command to the VEMMI application. The trigger events which induce the performance of local actions are:

- activation of a component;
- validation of a component.

The available report commands are:

- report OIN + CIN;
- report the current values of the component;
- report the values of all components in this object;
- report all values of all components in the parent objects that have changed (also via a VEMMI_Modify_component) since the last report or since the object creation if this is the first report.

Tables 36 and 37 show the possible trigger events and the possible reports for each component.

The overall structure of local actions is defined in 4.8.

TABLE 36/T.107

Trigger events

Component	Activation	Validation
Menu choice bar	✓	✓
Menu choice pull-down	✓	✓
Menu choice cascading	✓	✓
Menu choice pop-up	✓	✓
Menu choice separator		
Button	✓	✓
Separator		
Frame		
Text presentation area		
Text component		
Push button	✓	✓
Text input field	✓	✓
Check box	✓	✓
Radio button	✓	✓
List box	✓	✓
Combination box	✓	✓
Slider	✓	✓
Sensitive area	✓	✓
Sensitive text		
Graphic output area		

7.1.3 Relationship between objects and components

The closed state of an object overrules the opened state of a component. If an object is closed, the entire object including all its components is removed from the screen, regardless if its components are in the opened state or not. The opened state of an object does not overrule the closed state of a component. If an object gets opened, the components which are in the closed state are not displayed.

The inaccessible state parameter of an object overrules the accessible state parameter of a component. If an object is set inaccessible, the entire object including all its components becomes inaccessible, regardless if the components are accessible or not. The accessible state parameter of an object does not overrule the inaccessible state parameter of a component. If an object is set accessible, the user can only interact with those components that are accessible.

TABLE 37/T.107

Reports

Component	CIN	Value of all Components	Value (Note)
Menu choice bar	✓		
Menu choice pull-down	✓		
Menu choice cascading	✓		
Menu choice pop-up	✓		
Menu choice separator			
Button	✓		
Menu choice	✓		
Separator			
Frame			
Text presentation area			
Text component			
Push button	✓	✓	
Text input field	✓	✓	String
Check box	✓	✓	Boolean
Radio button	✓	✓	Boolean
List box	✓	✓	String list
Combination box	✓	✓	String
Slider	✓	✓	Integer, boolean
Sensitive area	✓	✓	
Sensitive text			
Graphic output area			
NOTE – The entries in this column specify the data type of the values that are reported.			

7.1.4 Open/close of audio, video, resource and metacode objects

If a sound object is opened, the sound shall be performed. At the end of the sound output, the object shall be considered closed.

The same rule applies to video objects. If the object is opened, the video display process starts and at its end the object is closed.

The open operation of a resource only checks if it is present at the terminal.

A metacode object is processed by the open operation. At the end of processing, it is considered closed.

7.1.5 Maximize operation

NOTE – The operation described in the following subclause is not mandatory for a VEMMI terminal. If a terminal does not implement this operation, it shall ignore the corresponding attributes and operations.

An object can be defined as maximizable. In this case it is displayed with a maximize button in its title row and upon the validation of this button it is displayed enlarged. The enlarged object is completed with a restore button and upon a validation of the restore button, the object is re-displayed in its initial size. These operations are terminal and UI dependent, especially the enlarged size. The host application only can mark an object as maximizable.

All object elements and components, except the Text Presentation Area, are enlarged on a terminal specific basis. The Text Presentation Area component of a Dialogue Box has a specific behaviour, in order to allow an enlarged object to add locally more text content in its display area. If the Text Presentation Area contains in its initial size the whole text, the maximize operation is terminal dependent. If the text is only partly displayed in its initial position, the terminal shall add additional text in the maximized object. The enlarged object can display the text in a new formatted form. This can be set by the host.

If the object is closed in its enlarged size and later reopened it is displayed in its initial (not enlarged) size. If a maximized object is deactivated it keeps its size.

Additional local operations on objects, which are not definable by the host can be offered to the user on a terminal specific basis. These can be: minimize, moving, iconize, unidirectional increase and decrease, etc.

7.1.6 Notational Conventions

For the next subclauses the following notational conventions apply:

- 1) In the following subclauses of this clause each object and component description contains an attribute list. These lists contain those attributes that are needed to describe the properties of the elements. The precise syntax of the element encoding is defined in clause 9.
- 2) The identification numbers for resource objects, BIN (Bitmap Identification Number), FIN (Font Identification Number), TIN (Text Identification Number), VIN (Videotex Identification Number) are special cases of OINs and used on the same level, e.g. as parameters of VEMMI commands. Different from them the CIN always denotes a component.

The naming conventions for the boolean attributes in the VEMMI element description follows the following rule:

- If the attribute is present in the encoding, its boolean value corresponds to the value expressed by its name.

Example

The default value of the state “Opened” is true. If the “closed” attribute is present in the encoding of a VEMMI element, the element shall be in the “Closed” state. If the “closed” attribute is not present in the encoding of a VEMMI element, the element shall be in its default state, the “Opened” state.

7.1.7 Mnemonic

When text data is used in a component, the text can contain the character “&” which marks the next character as a mnemonic key. The mnemonic key can be used to validate or activate the component. To present the “&” character in a text it shall be doubled (e.g.: text content = “&Cats && Dogs”; text presented to the user “Cats & Dogs”).

7.2 Text formats

7.2.1 VEMMI high quality text

The implementation of text attributes (change of text font, text size, etc. inside a text string) and the definition of sensitive areas for text (sensitive text or “hot-spot”) is realized with “In-text attributes”. These are applicable for the following elements:

- Text Presentation Area components of a Dialogue box.
- Labels.
 - 1) *Text attributes*

The attributed font can be changed inside a text output operation. The font is referenced via the FIN. A selected new font applies to all subsequent text up to the next in-text attribute which changes the font or to the end of the component. The initial attributed font applied to the first text of the component is defined as an additional attribute in the component.

The following example shows the use of in-text attributes. The brackets <...> denote attributes of the components with their parameter.

Example:

.....

<text1> = “Customer service note:(CR,LF,CR,LF)”

<use FIN 5>

<text2> = “110 size negatives”

<use FIN 6>

<text3> = “must not be cut but must be returned in strip form for reprinting.”

<use FIN 5>

<text4> = “If they are cut we are unable to make prints from them.”

<use FIN 7>

<text5> = “(CR,LF,CR,LF)Issued by the A.P.L.”

.....

FIN 5 is defined as ROMAN, height 12.

FIN 6 is defined as ROMAN, height 12, bold, underlined.

FIN 7 is defined as SWISS, height 8.

The above text sequence with in-text attributes might look as follows:

Customer service note: 110 size negatives <i>must not be cut but must be returned in strip form for reprinting.</i> If they are cut we are unable to make prints from them. Issued by the A.P.L.
--

2) *In-text attributes for the definition of sensitive text*

Each text string defined as sensitive has the reference to a Sensitive Text component. This component contains two attributes which define the activation and validation operations to be performed upon the user interaction with the Sensitive Text.

The two groups of attributes can be mixed in one text component, so that the application can apply text attributes, e.g. colour, to sensitive strings. Such a string may appear green and possibly in addition highlighted, in order to be easily distinguished from the plain text, which might be black on white. With these attributes, the application can offer to the user “hypertext”-like output with VEMMI.

7.2.2 Text labels and titles

The text contents of labels and titles are clipped at the borders of their containers. This shall be applied to the following attributes:

- all title attributes;
- all label attributes;
- text content attribute of Menu Choice components;
- list text attribute of List Box and Combination Box components.

7.3 The Application Bar

The Application Bar is subdivided into three different logical groups of Menu Choice components. These groups differ in their behaviour and functionality. The three different groups are named:

- Bar;
- Pull-Down Menu;
- Cascading Menu.

NOTE 1 – The vertical representation of the Application Bar is an optional feature for a VEMMI terminal. If the vertical attribute is present and set true in the encoding of an Application Bar, the terminal may ignore it and present the Application Bar horizontally. However, no error message shall be sent to the host.

NOTE 2 – All positioning and dimensioning attributes for an Application Bar and for all of its Menu Choice components are optional. If positioning or dimensioning attributes are defined in the encoding of an Application Bar, the terminal may ignore them and present the Application Bar starting on the top left corner of the DDA and dimension it according to the space requirements of the text content. However, no error message shall be sent to the host.

The Bar is a horizontal or vertical list of Menu Choice components. The Pull-Down Menus are vertical lists of Menu Choice components which are associated to the same Menu Choice component of the Bar. The Cascading Menus are vertical lists of Menu Choice, components which are associated to the same Menu Choice component of the Pull-Down Menu.

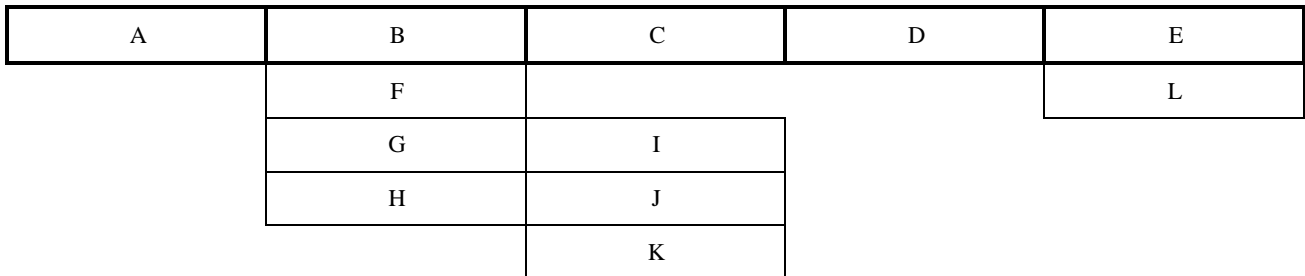
All Menu Choice components of the Application Bar shall only have text content.

A Menu Choice Separator component can be used to visually separate menu choice components in a logical group. It is defined by setting the attribute “Separator” with the definition of a regular Menu Choice.

The structure of the Application Bar Object is given, by the canonical descending order of its components (path left to right with priority to the depth).

In the following Application Bar example, the canonical descending order of the components is:

(AB(FG(IJK)H)CDE(L))

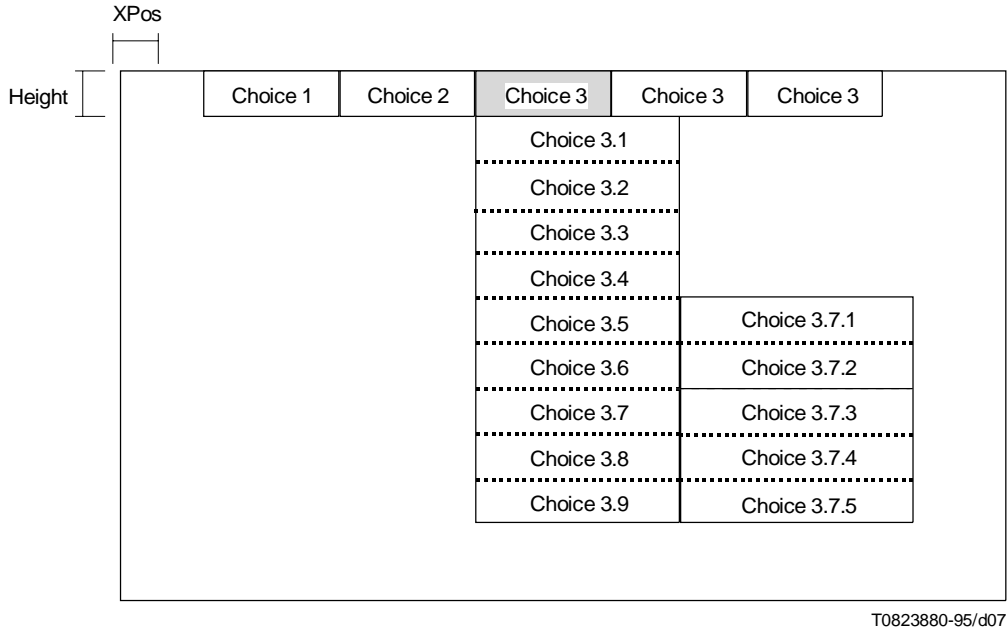


NOTE 3– The use of parenthesis in the above example is only to show the different levels of encapsulation of the different object groups for the path of description.

The description of the structure of the object is then:

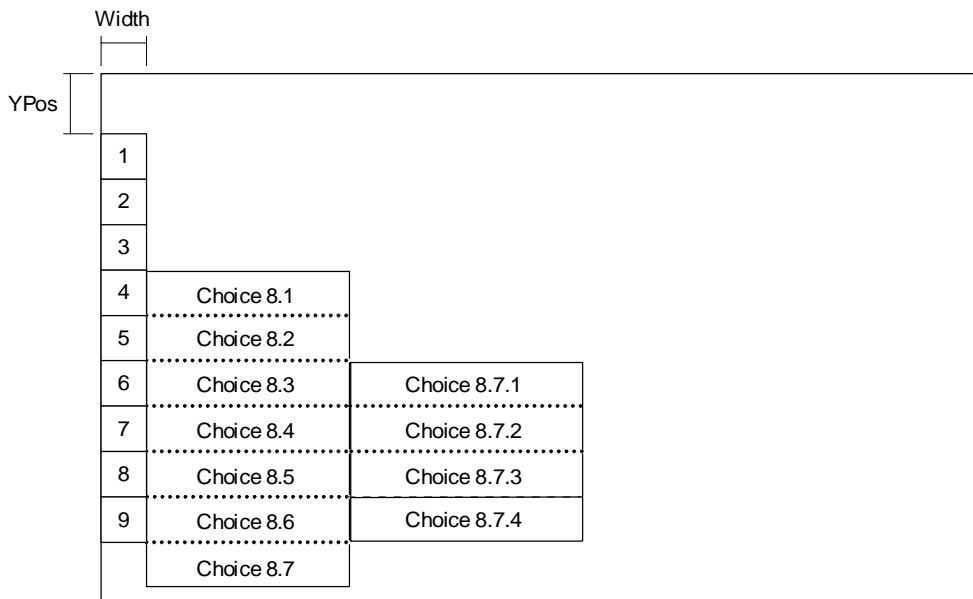
- Object: Application Bar.
- Component: Menu Choice Bar A.
- Component: Menu Choice Bar B.
- Component: Menu Choice Pull-Down F.
- Component: Menu Choice Pull-Down G.
- Component: Menu Choice Cascading I.
- Component: Menu Choice Cascading J.
- Component: Menu Choice Cascading K.
- Component: Menu Choice Pull-Down H.
- Component: Menu Choice Bar C.
- Component: Menu Choice Bar D.
- Component: Menu Choice Bar E.
- Component: Menu Choice Pull-Down L.

General Visual Aspect (see Figures 7 and 8)



T0823880-95/d07

FIGURE 7/T.107



T0823890-95/d08

FIGURE 8/T.107

Attributes

- Vertical: The Application Bar shall be presented vertical (see Note 1).
- XPos: This attribute carries the horizontal position of the element in NDC. It is only applicable to horizontal Application Bars (see Note 2).
- YPos: This attribute carries the vertical position of the element in NDC. It is only applicable to vertical Application Bars (see Note 2).
- Height: This attribute carries the height of the Bar. It is only applicable to horizontal Application Bars (see Note 2).
- Width: This attribute carries the width of the Bar. It is only applicable to vertical Application Bars (see Note 2).

- **FirstActive:** This attribute carries the CIN of the Menu Choice which is active by default, the first time the object is opened.
- **Closed:** The element shall be in the closed state.
- **NotAccessible:** The object is not accessible.

7.3.1 Composition

7.3.1.1 Menu Choice components of the Bar

Description

The Bar is a consecutive list of Menu Choices positioned side by side horizontally, or vertically.

Horizontal representation:

- the Application Bar shall be presented at the top of the DDA;
- the number of rows allocated for each Menu Choice shall be the same.

Vertical representation:

- the Application Bar shall be presented at the leftmost part of the DDA;
- the number of columns allocated for each Menu Choice shall be the same.

Behaviour

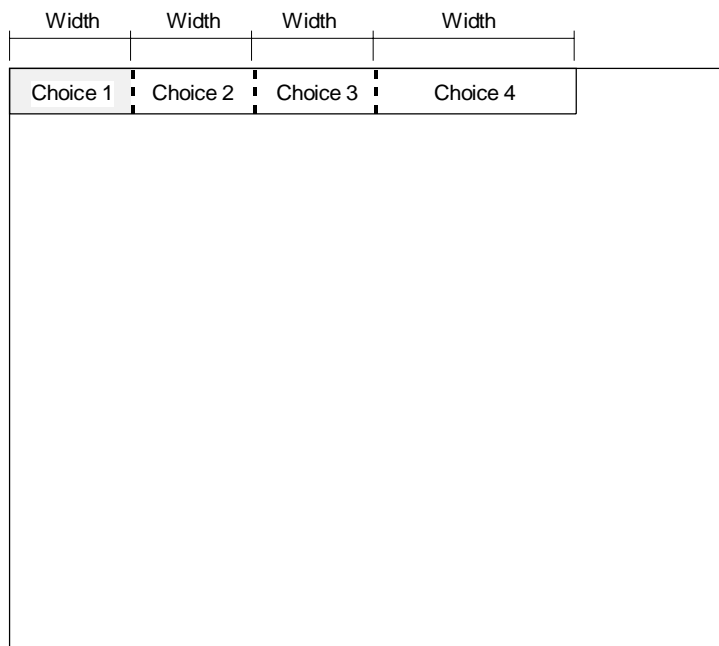
When the Application Bar is active, the terminal shall emphasize one Menu Choice, and offer shifting and validation facilities to the user. A mnemonic key can be used to validate the Menu Choice (see 7.1.7).

If a Menu Choice of the Bar is validated the associated local action shall be performed and then the associated Pull-Down Menu shall be opened (the Menu Choice remains emphasized).

Interactive functionality

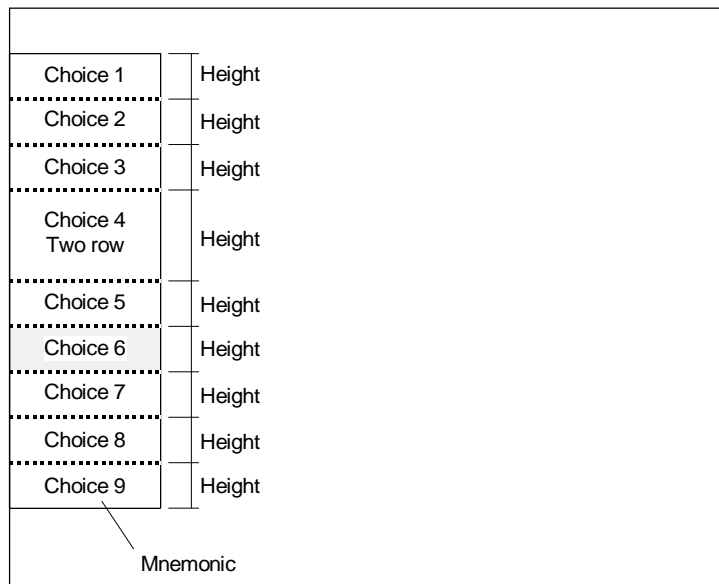
- shifting;
- validation.

Visual aspect (see Figures 9 and 10)



T0815510-94/d09

FIGURE 9/T.107
Horizontal menu bar



T0815520-94/d10

FIGURE 10/T.107
Vertical menu bar

Attributes

- **CIN:** This attribute carries the Component Identification Number.
- **Height:** This attribute carries the height of the component. It is only applicable to vertical Bars (see , 7.3, Note 2).
- **Width:** This attribute carries the width of the component. It is only applicable to horizontal Bars (see 7.3, Note 2).
- **NotAccessible:** The element shall not be accessible.
- **Text:** This attribute carries the text content of the component.
- **LocActAct:** This attribute carries the code for the local action which is associated to the component and triggered by its activation.
- **LocActVal:** This attribute carries the code for the local action which is associated to the component and triggered by its validation.

7.3.1.2 Menu Choice components of the Pull-Down Menu

Description

The Pull-Down Menu is a consecutive list of Menu Choices components associated to the same Menu Choice of the Bar and presented vertically on several rows.

The number of columns allocated for each Menu Choice shall be the same.

The Pull-Down Menu shall be positioned next to the associated Menu Choice of the Bar.

A Separator can be used in an Pull-Down Menu to visually separate different Menu Choices.

Behaviour

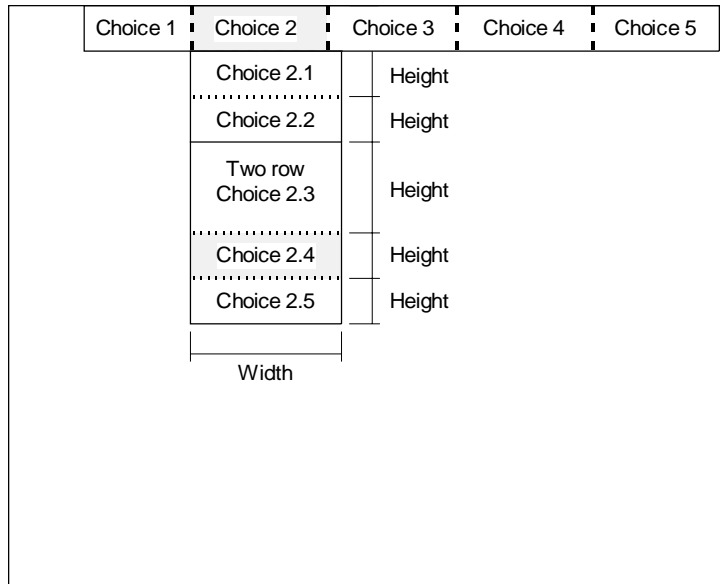
When the Pull-Down Menu is active, the terminal shall emphasize one Menu Choice, and offer to the user shifting and validation facilities. A mnemonic key can be used to validate the Menu Choice (see 7.1.7).

If a Menu Choice is validated the associated local action shall be performed and then the associated Cascading Menu shall be opened (the Menu Choice remains emphasized). If no Cascading Menu is associated to the Menu Choice after a user validation, the associated local action shall be performed and then the Pull-Down Menu shall be closed.

Interactive functionality

- shifting;
- validation.

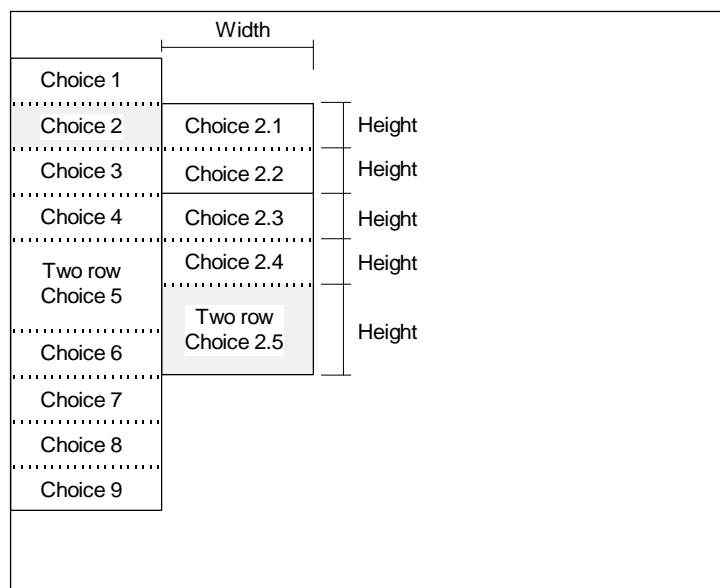
Visual aspect (see Figures 11 and 12)



T0824320-95/d11

FIGURE 11/T.107

Pull-down menu



T0824330-95/d12

FIGURE 12/T.107

Pull-down menu

Attributes

- CIN: This attribute carries the Component Identification Number.
- Height: This attribute carries the height of the component (see 7.3, Note 2).
- Width: This attribute carries the width of the component. It shall be equal to all Menu Choices of the Pull Down Menu (see 7.3, Note 2).
- NotAccessible: The element shall not be accessible.
- Text: This attribute carries the text content of the component.
- Separated: A Separator shall be drawn between the elements.
- LocActAct: This attribute carries the code for the local action which is associated to the component and triggered by its activation.
- LocActVal: This attribute carries the code for the local action which is associated to the component and triggered by its validation.

7.3.1.3 Menu Choice components of the Cascading Menu

Menu Choice components of Cascading Menu are common to the Pull-Down Menu and the Pop-Up Menu.

Description

The Cascading Menu is a consecutive list of Menu Choices associated to the same Menu Choice of the Pull-Down Menu or Pop-Up Menu and presented vertically on several rows.

The number of columns allocated for each Menu Choice shall be the same.

The Cascading Menu shall be positioned next to the associated Pull-Down Menu or Pop-Up Menu.

A Separator can be used in a Cascading Menu to visually separate different Menu Choices.

Figure 13 shows the recommended presentation for a Cascading Menu. If the terminal is not able to present a Cascading Menu like shown in Figure 13 it can, as a fallback solution, present the Menu Choices of the Cascading Menu in the Pull-Down Menu but it should visually associate the Menu Choices of the Cascading Menu to the corresponding Menu Choice of the Pull-Down Menu. The terminal should set the Menu Choice in the Pull-Down Menu, to which the Cascading Menu is associated, inaccessible and present it as a title for the Menu Choices of the Cascading Menu (see Figure 14).

Behaviour

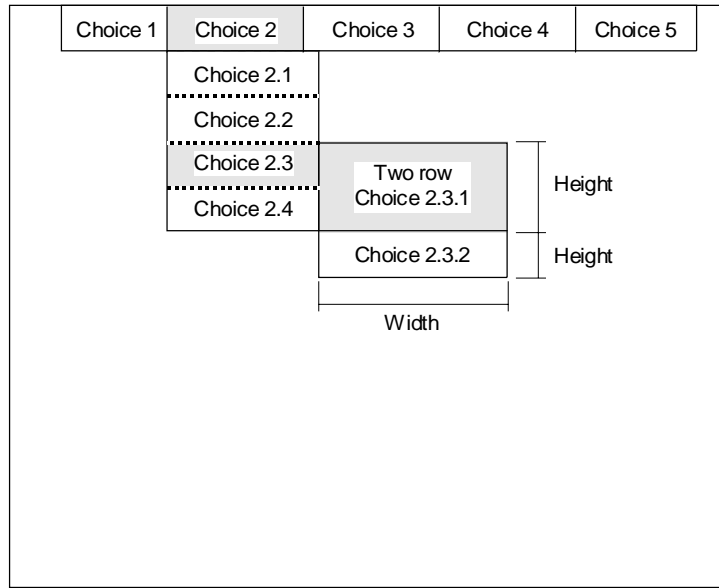
When the Cascading Menu is active, the terminal shall emphasize one Menu Choice, and offer to the user shifting and validation facilities. A mnemonic key can be used to validate the Menu Choice (see subclause 7.1.7).

After the validation of a Menu Choice, the associated local action shall be performed and then the Cascading Menu and the Pull-Down Menu (Pop-Up Menu) shall be closed.

Interactive functionality

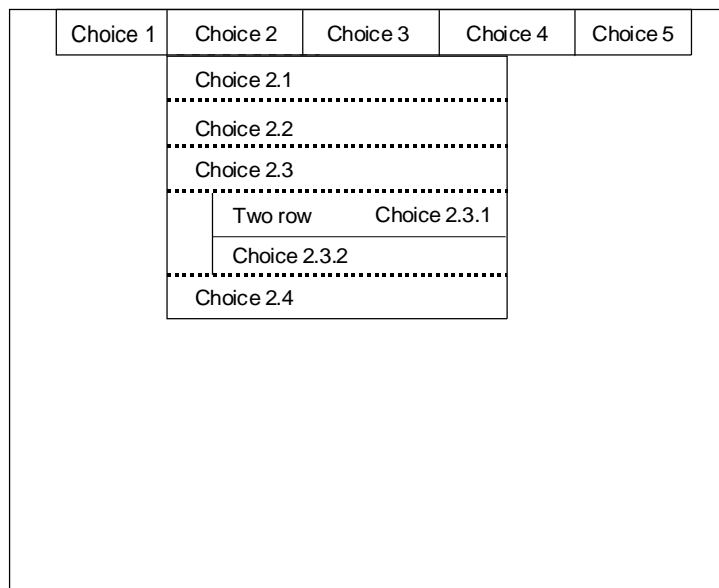
- shifting;
- validation.

Visual aspect (see Figure 13 and 14)



T0823920-95/d13

FIGURE 13/T.107
Cascading menu (recommended presentation)



T0823930-95/d14

FIGURE 14/T.107
Cascading menu (fallback presentation)

Attributes

- **CIN:** This attribute carries the Component Identification Number.
- **Height:** This attribute carries the height of the component (see 7.3, Note 2).
- **Width:** This attribute carries the width of the component. It shall be equal to all Menu Choices of the Cascading Menu (see 7.3, Note 2).
- **NotAccessible:** The element shall not be accessible.

- Text: This attribute carries the text content of the component.
- Separated: A Separator shall be drawn between the elements.
- LocActAct: This attribute carries the code for the local action which is associated to the component and triggered by its activation.
- LocActVal: This attribute carries the code for the local action which is associated to the component and triggered by its validation.

7.4 The Button Bar

Description

The Button Bar is a consecutive list of Buttons positioned side by side horizontally, or vertically.

Horizontal representation:

- the number of rows allocated for each Button shall be the same.

Vertical representation:

- the number of columns allocated for each Button shall be the same.

NOTE – If the “Height” and the “Width” attribute are present in the definition of a Bar all Buttons in the Bar shall have the same dimensions. The dimensioning attributes for the components of the Button Bar shall then not be present. If only the “Height” attribute is present in the definition of a horizontal Button Bar, it denotes the height for the entire object. The width for the components is then given with each component. If only the “Width” attribute is present in the definition of a vertical Button Bar it denotes the width for the entire object. The height for the components is then given with each component.

Behaviour

When the Button Bar is active, the terminal shall emphasize one Button, and offer to the user shifting and validation facilities.

Interactive functionality

- shifting;
- validation.

Visual aspect (see Figure 15)

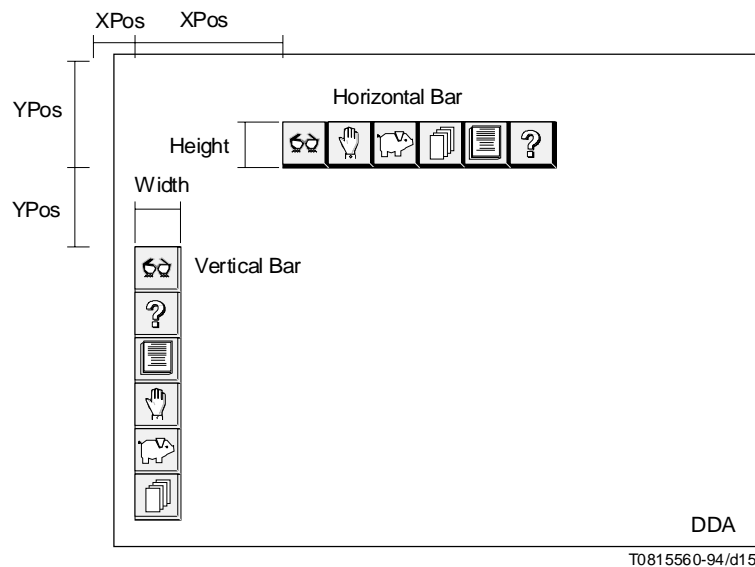


FIGURE 15/T.107
Button bar

Attributes

- XPos: This attribute carries the horizontal position of the element in Normalized Device Coordinate (NDC).
- YPos: This attribute carries the vertical position of the element in NDC.
- Vertical: Boolean switch between the alternatives.
- Height: This attribute carries the height of the Button Bar. It is only applicable to horizontal Button Bars (see Note).
- Width: This attribute carries the width of the Button Bar. It is only applicable to vertical Button Bars (see Note).
- FirstActive: This attribute carries the CIN of the Button which is active by default, the first time the object is opened.
- Modal: The element shall be modal.
- Closed: The element shall be in the closed state.
- NotAccessible: The element shall not be accessible.

7.4.1 Composition

7.4.1.1 The Button component

Description

The Button is a rectangular area in the DDA.

If formed with text data it should be represented by a text label, possibly associated with a graphic to draw the shape of the button. The button graphic shape shall be included in the space allocated by the VEMMI application for the whole Button component. If the graphic shape does not cover the entire space allocated for the button, the terminal can resize the content. The terminal may centre the text data in the element display area reserved by the VEMMI application for the component.

The drawing of the Button and the possible visual trigger effect are terminal dependent.

Behaviour

When a Button is active, the terminal shall emphasize it. A mnemonic key can be used to validate the Button (see 7.1.7).

When a trigger effect is implemented, the Button shall go back to the initial display state after user validation.

Interactive functionality

- activation;
- validation.

Visual aspect (see Figure 16)

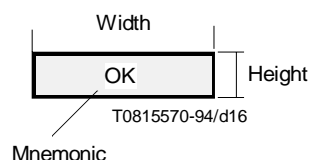


FIGURE 16/T.107

Button

Attributes

- CIN: This attribute carries the Component Identification Number.
- Height: This attribute carries the height of the component. It is only applicable to Buttons of vertical Button Bars (see 7.4, Note).
- Width: This attribute carries the width of the component. It is only applicable to Buttons of horizontal Button Bars (see 7.4, Note).
- Closed: The element shall be in the closed state.
- NotAccessible: The element shall not be accessible.
- BIN: The BIN of the component.
- Text: The text content of the component.
- LocActAct: This attribute carries the code for the local action which is associated to the component and triggered by its activation.
- LocActVal: This attribute carries the code for the local action which is associated to the component and triggered by its validation.

7.5 The Pop-Up Menu

The Pop-Up Menu is subdivided into two different logical groups of Menu Choice components. These groups differ in their behaviour and functionality. The two different groups are named:

- Primary Pop-Up Menu;
- Cascading Menu.

The Primary Pop-Up Menu is a vertical list of Menu Choice components. The Cascading Menu is a vertical list of Menu Choice components associated to the same Menu Choice component of the Primary Pop-Up Menu.

A Menu Choice Separator can be used to visually separate menu choice components in a logical group. It is defined by setting the attribute “Separator” with the definition of a regular Menu Choice.

The structure of the Pop-Up Menu Object is given, with the following conventions, by the canonical descending order of its components (path top down).

In the following Pop-Up Menu example, the canonical descending order of the components is:

(BFG(IJK)H)

B	
F	
G	I
H	J
	K

NOTE 1 – The parenthesis in the above example are only for showing the different levels of encapsulation of the different object groups for the path of description.

The description of the structure of the object is then:

- Object: Pop-Up Menu.
- Component: Menu Choice Pop-Up B.
- Component: Menu Choice Pop-Up F.
- Component: Menu Choice Pop-Up G.
- Component: Menu Choice Cascading I.

- Component: Menu Choice Cascading J.
- Component: Menu Choice Cascading K.
- Component: Menu Choice Pop-Up H.

Interactive functionalities

- shifting;
- validation.

General visual aspect (see Figure 17)

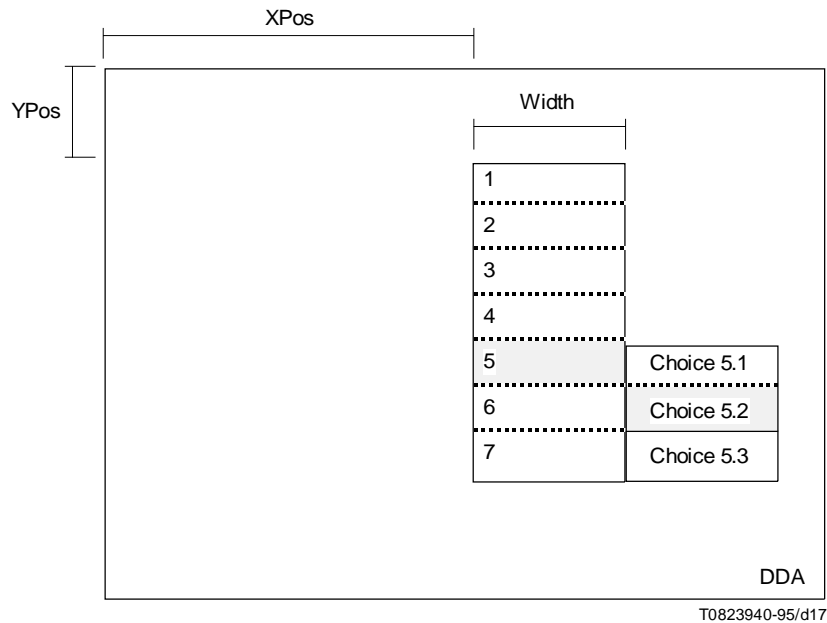


FIGURE 17/T.107

A text title may be given to the Pop-Up Menu which shall be displayed in its first row of the Primary Pop-Up Menu.

NOTE 2 – All dimensioning attributes for the Menu Choice components of the Pop-Up Menu are optional. If dimensioning attributes are defined in the encoding of a Pop-Up Menu, the terminal may ignore them and dimension it according to the space requirements of the text content. However, no error message shall be sent to the host.

Attributes

- XPos: This attribute carries the horizontal position of the element in NDC.
- YPos: This attribute carries the vertical position of the element in NDC.
- Width: This attribute carries the width of the Pop-Up Menu (see Note 2).
- Title: This attribute carries the title of the object. The title shall be displayed in the first row of the object.
- TitleFont: This attribute carries the FIN of the title.
- FirstActive: This attribute carries the CIN of the component which is active by default, the first time the object is opened.
- Modal: The element shall be modal.
- Closed: The element shall be in the closed state.
- NotAccessible: The element shall not be accessible.

7.5.1 Composition

7.5.1.1 Menu choice components of the primary pop-up menu

Description

The Primary Pop-Up Menu is a consecutive list of Menu Choices and presented vertically on several rows.

The number of columns allocated for each Menu Choice shall be the same.

A Separator can be used in a Primary Pop-Up Menu to visually separate different Menu Choices.

Behaviour

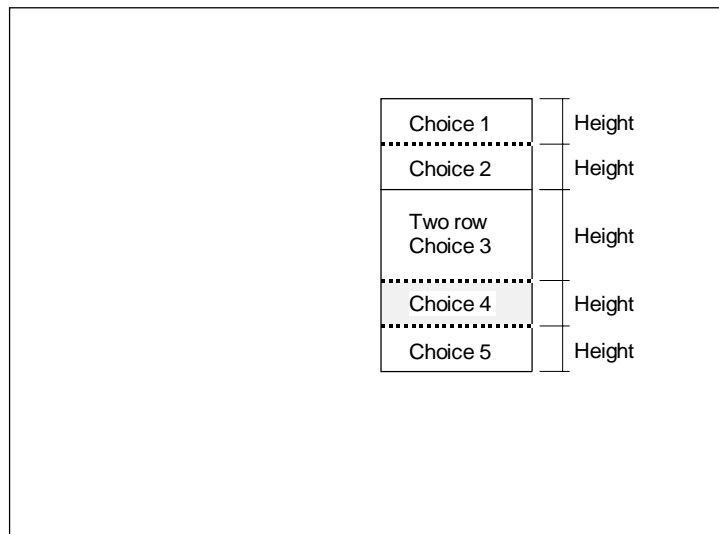
When the Primary Pop-Up Menu is active, the terminal shall emphasize one Menu Choice, and offer to the user shifting and validation facilities. A mnemonic key can be used to validate the Menu Choice (see 7.1.7).

If a Pop-Up Menu component is validated the associated local action shall be performed and then the associated Cascading Menu shall be opened (the Pop-Up Menu component remains emphasized). If no Cascading Menu is associated to the Menu Choice after a user validation, the associated local action shall be performed and then the Pop-Up Menu shall be closed.

Interactive functionality

- shifting;
- validation.

Visual aspect (see Figure 18)



T0823950-95/d18

FIGURE 18/T.107

Pop-up menu

Attributes

- CIN: This attribute carries the Component Identification Number.
- Height: This attribute carries the height of the component (see 7.5, Note 2).
- Closed: The element shall be in the closed state.

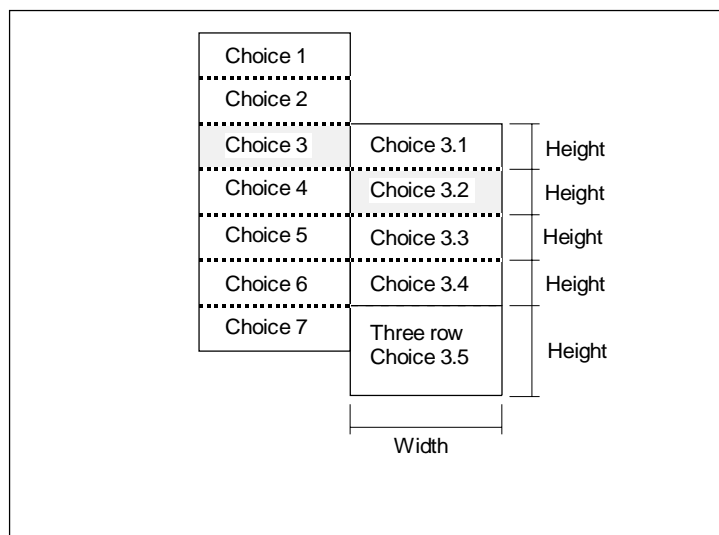
- **NotAccessible:** The element shall not be accessible.
- **Text:** This attribute carries the text content of the component.
- **Separated:** The elements shall be separated.
- **LocActAct:** This attribute carries the code for the local action which is associated to the component and triggered by its activation.
- **LocActVal:** This attribute carries the code for the local action which is associated to the component and triggered by its validation.

7.5.1.2 Menu choice components of the cascading menu

Description, behaviour, interactive functionality, attributes

See 7.3.1.3.

Visual aspect (see Figure 19)



T0823960-95/d19

FIGURE 19/T.107

Recommended presentation of a pop-up menu with associated cascading menu

7.6 The Dialogue Box

Description

The Dialogue Box is a rectangular area in the DDA which contains VEMMI components in order to establish user interaction.

The components themselves contain (or have references to) other components or resource objects. Figure 20 shows the relationship. Each arrow indicates a possible reference to a component or a resource object. These references can be multiple as well as the dialogue box can contain several components of the same type.

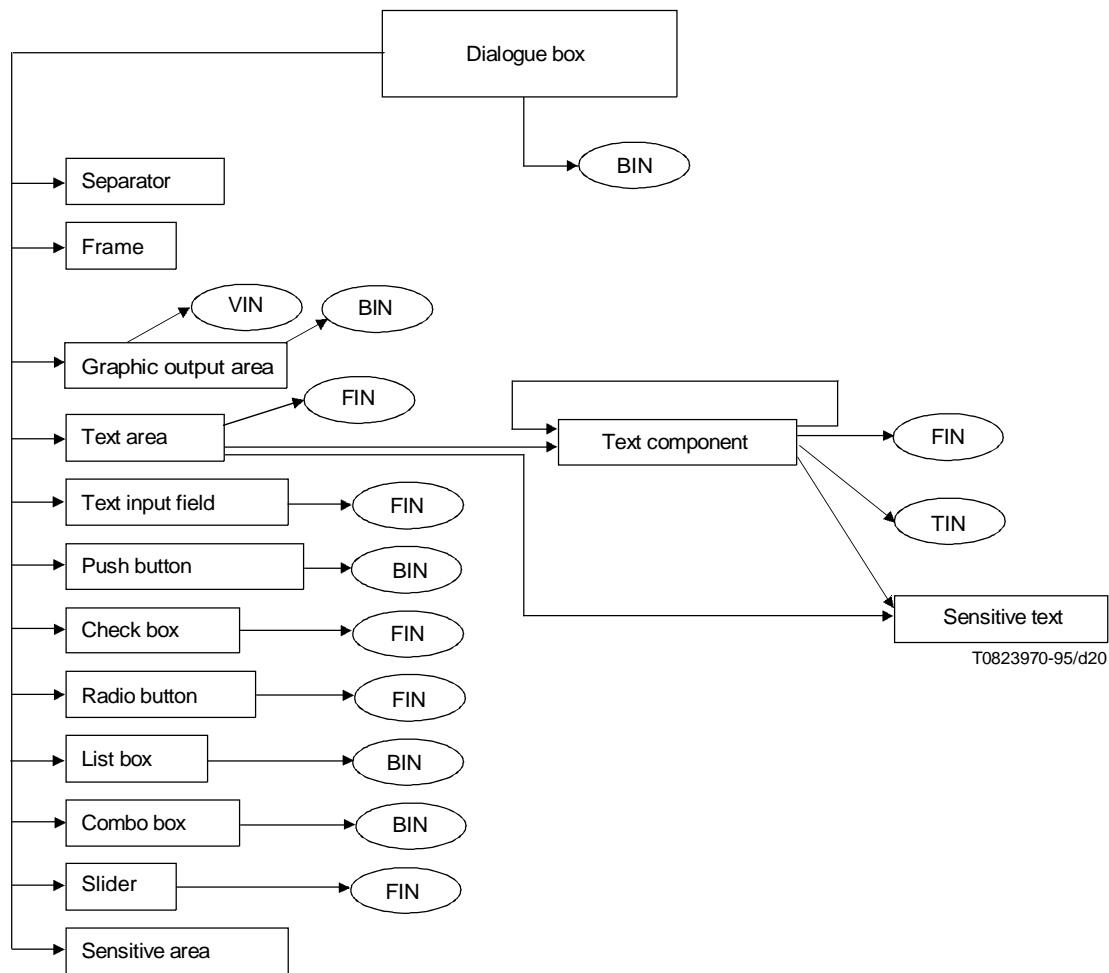


FIGURE 20/T.107
Element relationships within the dialogue box

A text title may be given to the Dialogue Box which shall be displayed in its first row. The title height equals 1/32 NDC.

The Dialogue Box can be provided with a border area which should be equal to one character position. When present, this border shall be included in the dimensions defined by the VEMMI application. When a border is requested, a frame shall be drawn by the terminal. This frame drawing is terminal dependent (it should not be wider than 1/160 NDC).

The object origin of the Dialogue Box is the top left corner of the box rectangle, independent whether the box has a title or not. The components are positioned relatively to this point for Dialogue Boxes that do not have a text title. In the case of boxes with a text title, this point is moved downwards by a distance corresponding to the titles height and the borders width and horizontally by the distance equal to the border width.

The background area of the Dialogue Box is the rectangle defined by the box dimensions, except the title bar and the border. The background can be coloured uniform or with a data content.

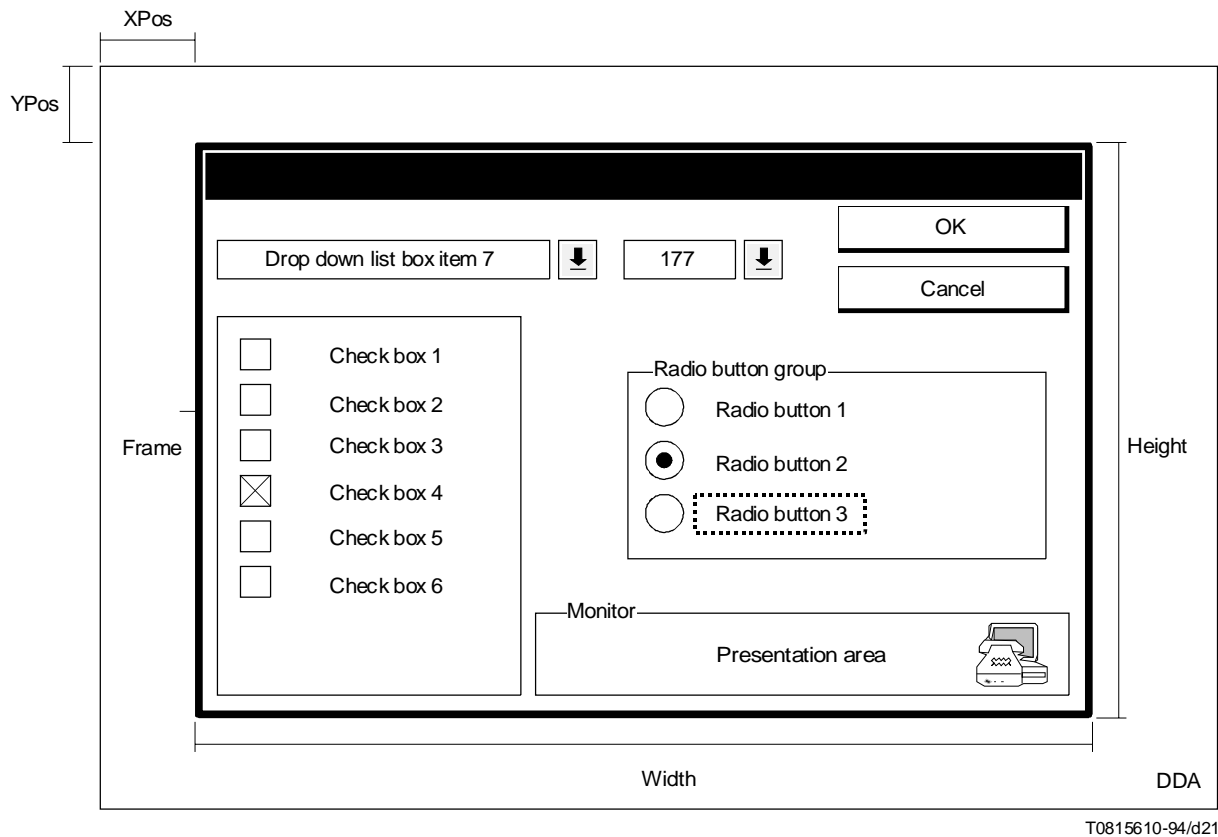
Behaviour

The user can activate any component with a pointing device or with the keyboard. If the user activates the components with the keyboard keys providing the functionality “Previous” and “Next”, the order of activation should correspond to the reception order of the components of the Dialogue Box.

Interactive functionality

- Moving.

Visual aspect (see Figure 21)



T0815610-94/d21

FIGURE 21/T.107

Dialogue box

Attributes

- XPos: This attribute carries the horizontal position of the element in NDC.
- YPos: This attribute carries the vertical position of the element in NDC.
- Width: This attribute carries the width of the object.
- Height: This attribute carries the height of the object.
- NoBorder: No border shall be drawn.
- Title: This attribute carries the title of the object. The title shall be displayed in the first row of the object.

- **FirstActive:** This attribute carries the CIN of the component which is active by default, the first time the object is opened.
- **Modal:** The element shall be modal.
- **Closed:** The element shall be in the closed state.
- **NotAccessible:** The element shall not be accessible.
- **StoreInitialValues:** The terminal shall store the initial values of the component at the time of the object creation. If a local action “restore initial values” is applied, these stored values shall be restored.
- **Maximizable:** The object is maximizable.
- **Colour:** Colour index of the background.
- **BIN:** Bitmap identification number to fill the Dialogue Box Background.
- **DispType:** Specifies the display mode of the background bitmap: centred, stretched (default) or tiled.

7.6.1 Composition

7.6.1.1 The Separator component

Description

The Separator is either a horizontal or a vertical solid line used to separate different areas within a Dialogue Box.

Behaviour

The Separator shall be inaccessible.

Interactive functionality

- None.

Visual aspect (see Figure 22)

Attributes

- **CIN:** This attribute carries the Component Identification Number.
- **XPos:** This attribute carries the horizontal position of the element in NDC.
- **YPos:** This attribute carries the vertical position of the element in NDC.
- **Vertical:** The element shall be drawn vertically.
- **Height:** This attribute carries the height of the component. It is only applicable to vertical Separators.
- **Width:** This attribute carries the width of the component. It is only applicable to horizontal Separators.
- **Closed:** The element shall be in the closed state.
- **Colour:** Colour index.

7.6.1.2 The Frame component

Description

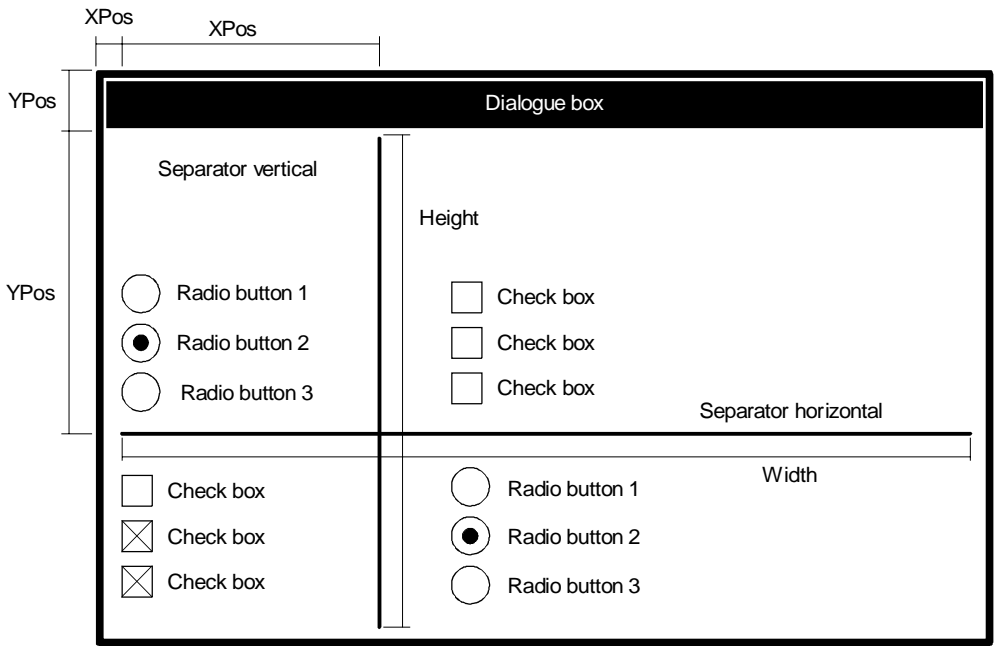
A Frame consists of four solid lines which visually separate a rectangular area of the Dialogue Box.

Behaviour

The Frame shall be inaccessible.

Interactive functionality

- None.

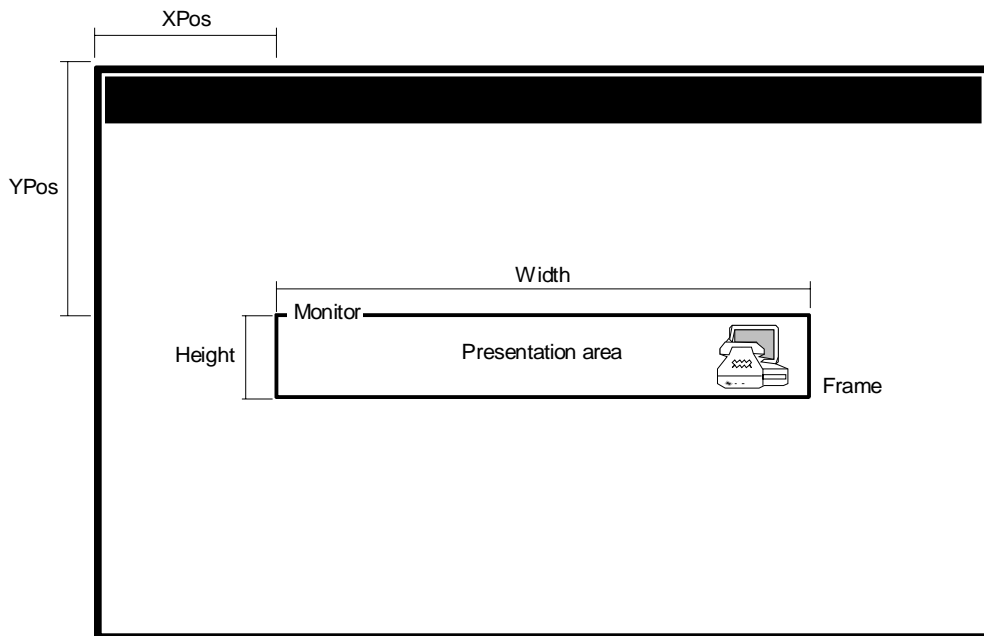


T0824340-95/d22

FIGURE 22/T.107

Separator

Visual aspect (see Figure 23)



T0815630-94/d23

FIGURE 23/T.107

Frame titled with a text presentation area

Attributes

- CIN: This attribute carries the Component Identification Number.
- XPos: This attribute carries the horizontal position of the element in NDC.
- YPos: This attribute carries the vertical position of the element in NDC.
- Width: This attribute carries the width of the component.
- Height: This attribute carries the height of the component.
- Closed: The element shall be in the closed state.
- Colour: Colour index.

7.6.1.3 The Text Presentation Area component

Description

The Text Presentation Area is an area intended to present text data to the user. Basically, this element is used to present text that fits in one window (the space defined by one text area component) or to present large text data where the user has to use scrolling tools in order to navigate through the text.

The Text Presentation Area has its starting location in the top left corner of the rectangle defined by the component and the text is displayed row by row from the left to the right. The text content can contain “In-Text” attributes to switch between different attribute fonts and to define “sensitive text strings”. The “In-Text” attributes can appear multiple, in any order, and they are evaluated during the display operation sequentially and on the basis of structural information contained in text components (“previous, next” attributes).

Scrolling, Border, Title

If the space needed for the presentation of the data is bigger than the space allocated for the component, vertical scrolling tools shall be provided by the terminal application. The terminal may present:

- scrolling tools next to the right side of the area;
- reduced scrolling tools as buttons overlapping the area;
- cursor keys that provide the scrolling functionality.

The space needed for the scrolling tools is included in the overall dimensions of the component.

The component can be provided with a border area. When present, the border shall be included in the overall dimensions of the component. When a border is requested a frame shall be drawn by the terminal. The frame drawing is terminal dependent. It should not be wider than 1/160 NDC. If scrolling tools are provided next to the right side of the component, the border shall be extended to include them visually in the box.

Maximize operation

The size of the Dialogue Box can be changed by the user with the scrolling tools provided by the local Graphical User Interface (GUI) and via a defined maximize button. The recommended strategy is described in 7.1.5. These are default recommendations. If the attribute “NoFormat” is selected for the component, the following applies:

- a vertical increase should not change the text form, no additional text should be added;
- a vertical decrease should clip the text on the new border;
- a horizontal increase should add new text lines;
- a horizontal decrease should clip the text.

Interactive functionality

- Activation and validation of sensitive text, defined via in-text attributes.

NOTE – To transfer a large amount of text data, it is recommended not to use the direct text definition but to use the Text Resource object and transmit the data using the VEMMI resource transfer.

Visual aspect (see Figure 24)

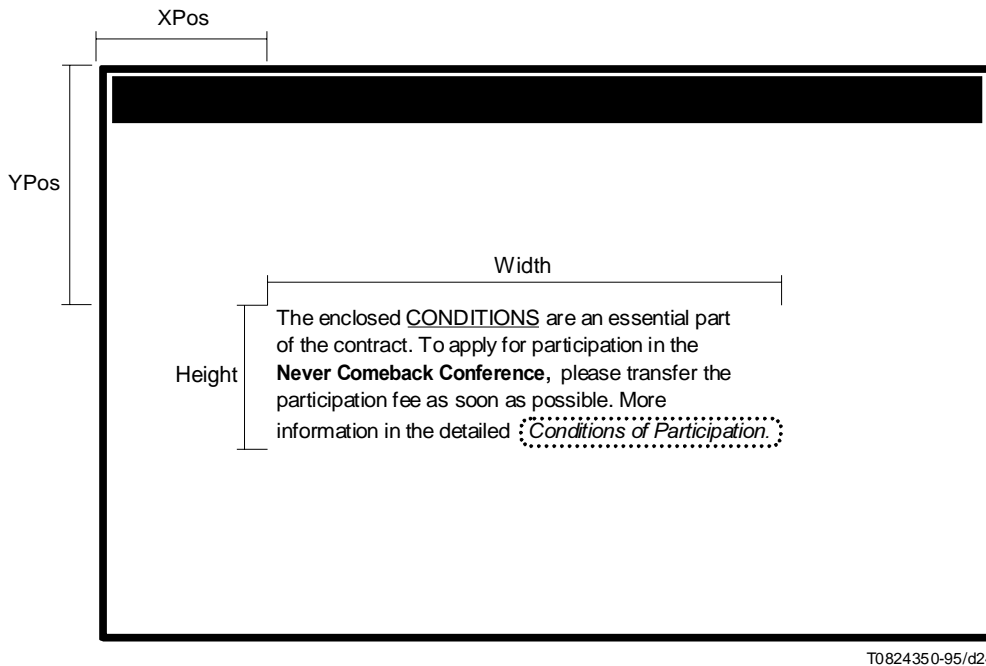


FIGURE 24/T.107
Text presentation area

Attribute

- CIN: This attribute carries the Component Identification Number.
- XPos: This attribute carries the horizontal position of the element in NDC.
- YPos: This attribute carries the vertical position of the element in NDC.
- Width: This attribute carries the width of the component.
- Height: This attribute carries the height of the component.
- NoScrollingTools: No scrolling tools shall be drawn by the terminal.
- Format: A maximization of the dialogue box shall use the limited formatting rules described above.
- NoBorder: No Border shall be drawn to frame the element.
- InitialFnt: FIN of the attribute font to be applied on the first text.

- **Autoscroll:** A modification of a component referenced by the text area leads to an immediate display of this modified component.
- **InText:** Text content possibly including in-text attributes, and references to font objects.
- **TextCompRef:** Reference to a text component.
- **Closed:** The element shall be in the closed state.

7.6.1.4 The Text component

Description

This component is used to split large text data in units (text components) and to define the necessary structural information (concatenation of the text components) in order to be displayed in a text area. This structural information are references to the previous and next text components, defined with the attributes “NextText” and “PreviousText”.

Display Concept

The components are not necessarily of the same length. During the display of concatenated text components, the user should have the impression of a continuous text. The continuity should not be interrupted by a switch from one component to the other. Vertical scrolling capabilities, on a row-by-row and page-by-page basis, shall be offered within consecutive components. This display continuity is interrupted in two cases:

- A component contains no further concatenation, no “NextText” during a forward scrolling or no “PreviousText” during a backward scrolling. The display does not continue because the just displayed text pertains to a topic which has reached its logical end. Figure 29 shows two sequences of concatenated components: CINs 7,8,9 and CIN 17.
- “Jumps” due to the validation of sensitive text as part of the displayed components or other interactive components of the dialogue box. Sensitive text attribute can contain as a local action an element specific command, e.g. “open component of parent object”. The referenced parent component is a text component which will be displayed in the starting position and not consecutively to the last displayed component. This effect is shown in Figures 27 and 28.

The text to be displayed is contained in the component definition, in another referenced text component or in a referenced text resource object. If such a resource object is not available in the terminal, it can be requested from the host application with the command VEMMI_Object_Retransmission and the Rule B shall be applied (see 5.3.2). Such a request might be initiated due to a forward scroll to data which is not yet in the terminal, but in general there is no direct relation between the scrolling action and the object retransmission request, because the terminal keeps the already received objects.

NOTE – To transfer a large amount of text data, it is recommended not to use the direct text definition but to use the Text Resource object and transmit the data using the VEMMI resource transfer.

Visual aspect (see Figures 25 to 28)

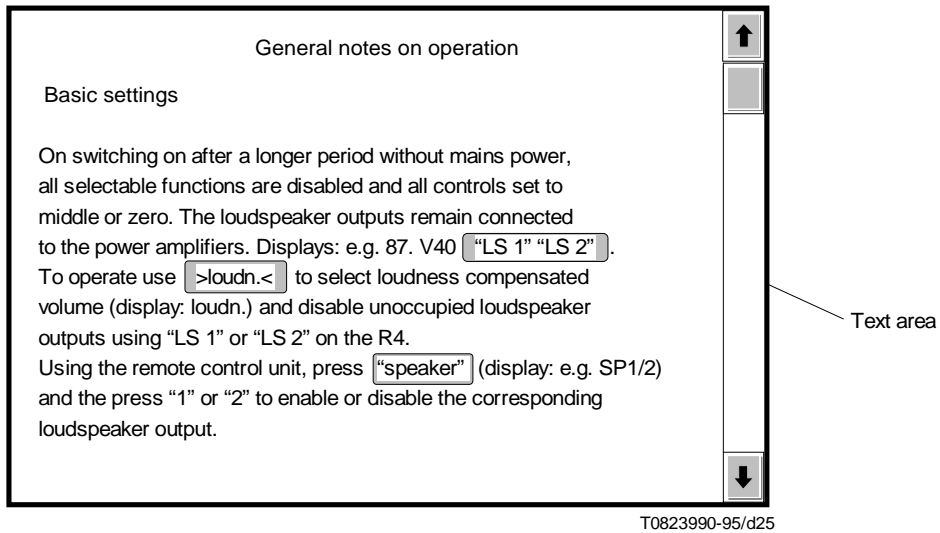


FIGURE 25/T.107

A text area with text component(s), scroll bars and sensitive texts (Attribute "NoFormat" is not set). Now, the user becomes active and scrolls downward ...

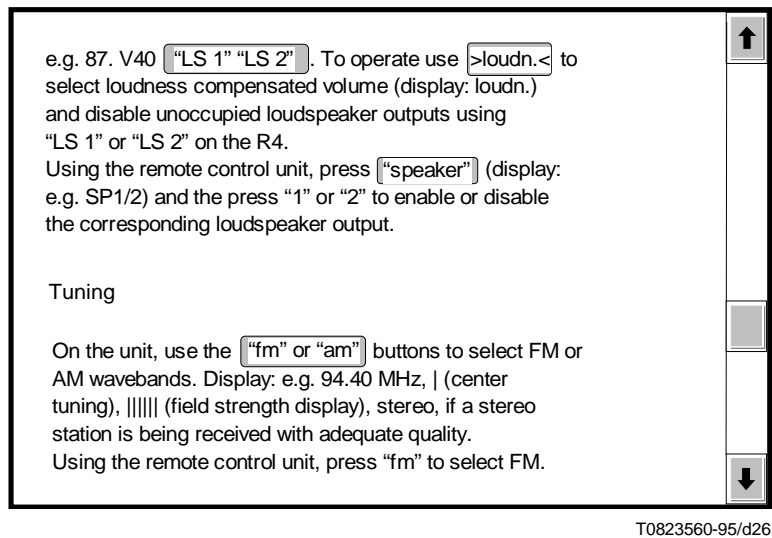
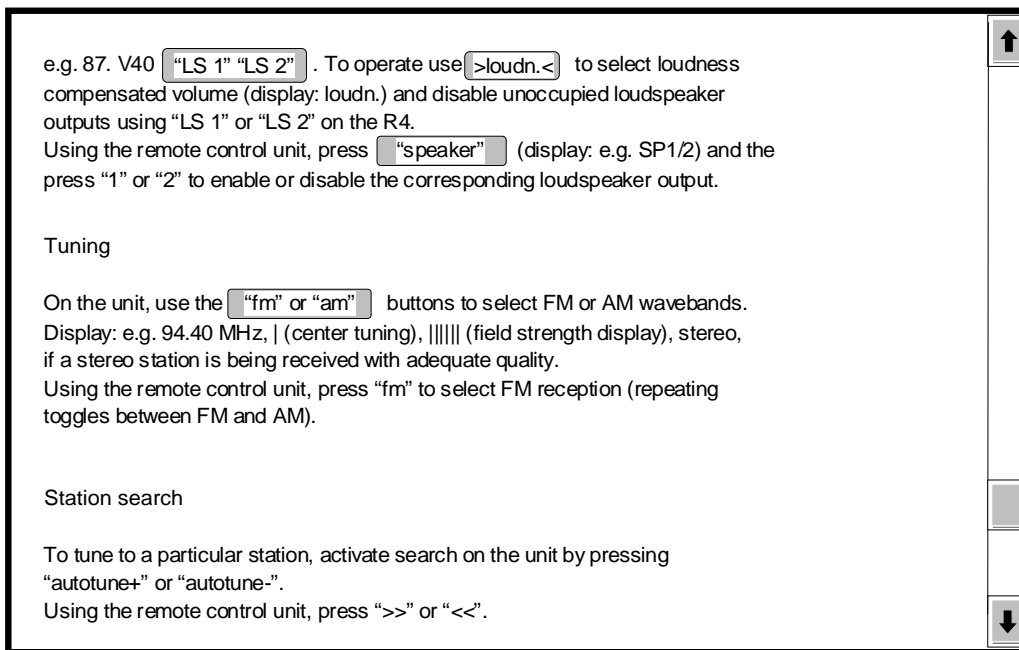


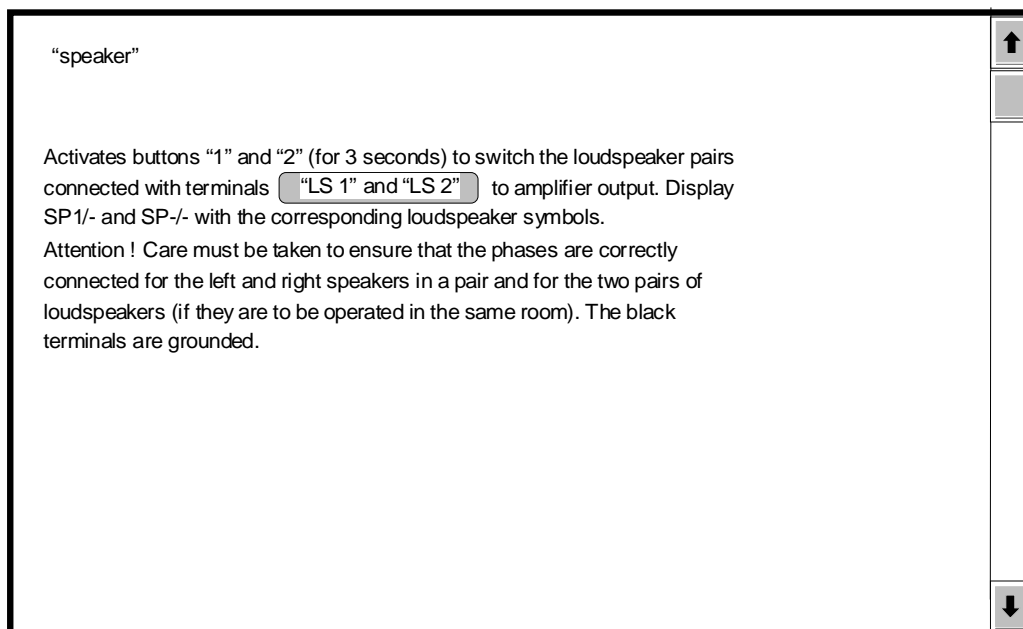
FIGURE 26/T.107

... in addition he/she enlarges the window on both directions, ...



T0823570-95/d27

FIGURE 27/T.107
... and finally, clicks on the hot spot "speaker"



T0823580-95/d28

FIGURE 28/T.107
The "speaker" text is displayed

A possible component architecture for the display sequence shown in Figures 25 to 29 can be the following (the in-text attributes are not listed):

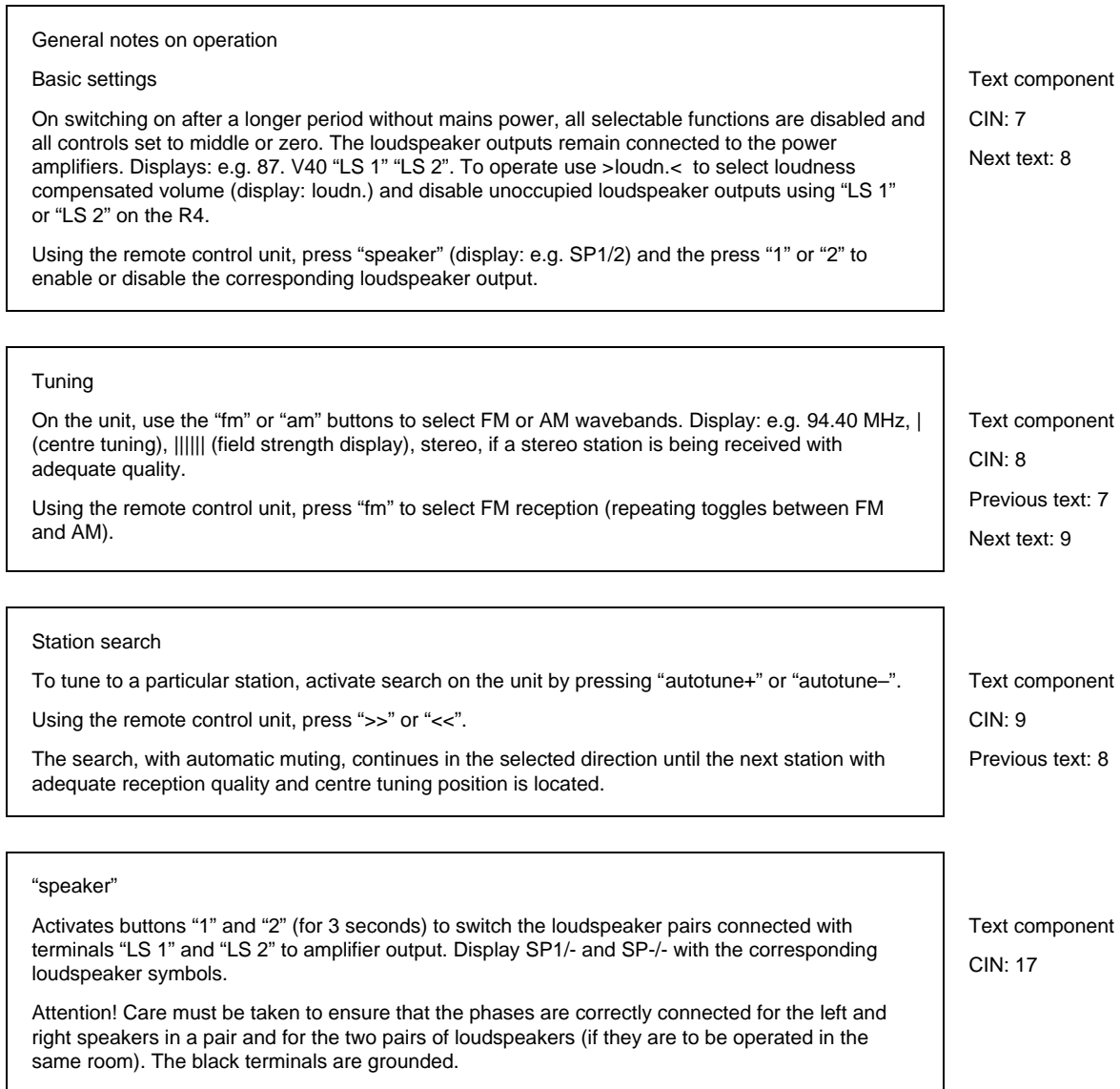


FIGURE 29/T.107

Text component architecture

The local interactivity can be implemented by the correct cooperation of the local actions (LocActVal parameter) which are part of the sensitive text and other interactive elements of the dialogue box. In the example case, the user can skip from the text of Figure 27 to Figure 25 with the Sensitive Text "Speaker" if the Sensitive Text contains the local action "open component 7".

Attributes

- CIN: Component Identification Number.
- PreviousText: CIN of the backward concatenated text component.
- CurrentText: Text definition, including references to font resource objects, text resource objects and sensitive text components.
- NextText: CIN of the forward concatenated text component.

7.6.1.5 The Sensitive Text Component

These components define the activation and validation operations for sensitive text strings. These are part of text components.

Visual aspect

See 7.6.1.4.

Attributes

- CIN: Component Identification Number.
- NotAccessible: The element shall not be accessible.
- LocActAct: Specifies the local actions which are associated to the component and triggered by its activation.
- LocActVal: Specifies the local actions which are associated to the component and triggered by its validation.

7.6.1.6 The Graphic Output Area component

Description

The Graphic Output Area is a rectangle to display graphical data (bitmaps, videotex data, etc.). A bitmap is referenced via a BIN. Three different display modes exist: the bitmap can be centred in the Graphic Output Area, it can be stretched to cover the whole area or it can be tiled. In any case the bitmap shall not exceed the space reserved with this component. If the referenced bitmap does not exist while the component is opened, it has no visual effect.

Videotex data are referenced via a VIN. In any case the Videotex data shall not exceed the space reserved with this component. If the referenced Videotex data content does not exist while the component is opened, it has no visual effect.

Behaviour

It is inaccessible.

Interactive functionalities

- None.

Visual aspect (see Figure 30)

Attributes

- CIN: Component Identification Number.
- XPos: Horizontal position of the top left corner of the bitmap rectangle.
- YPos: Vertical position of the top left corner of the bitmap rectangle.
- Width: Width of the bitmap rectangle.
- Height: Height of the bitmap rectangle.
- Closed: The element shall be in the closed state.
- DispType: Specifies the display mode of the bitmap: centred, stretched (default) or tiled.
- BIN: Bitmap Identification Number.
- VIN: Videotex Identification Number.

7.6.1.7 The Push Button component

Description, behaviour, interactive functionality

See 7.4.1.1.

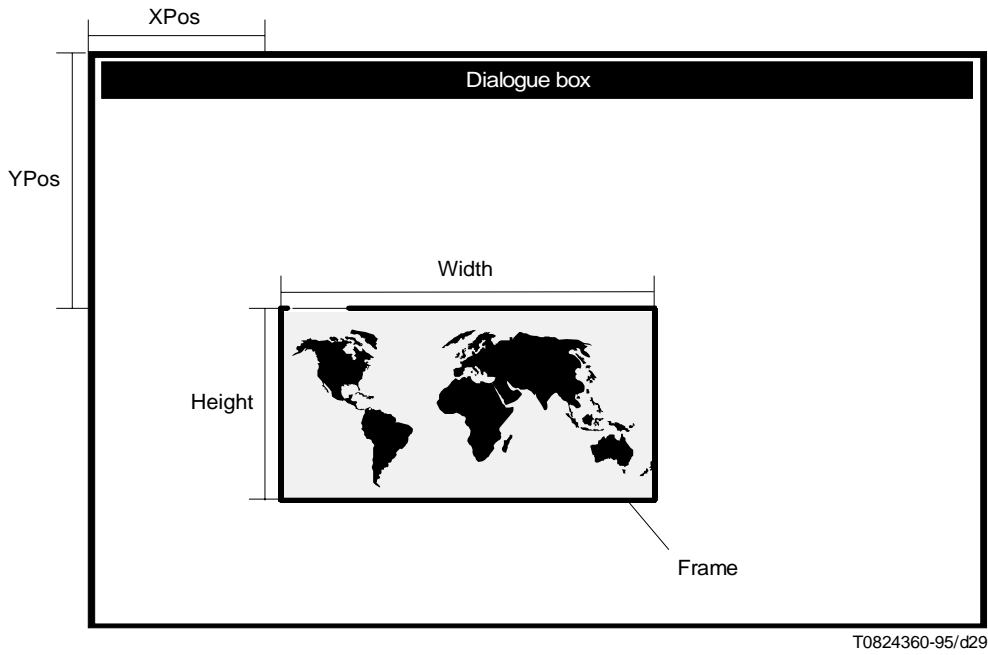


FIGURE 30/T.107
Frame and a bitmap

Visual aspect (see Figure 31)

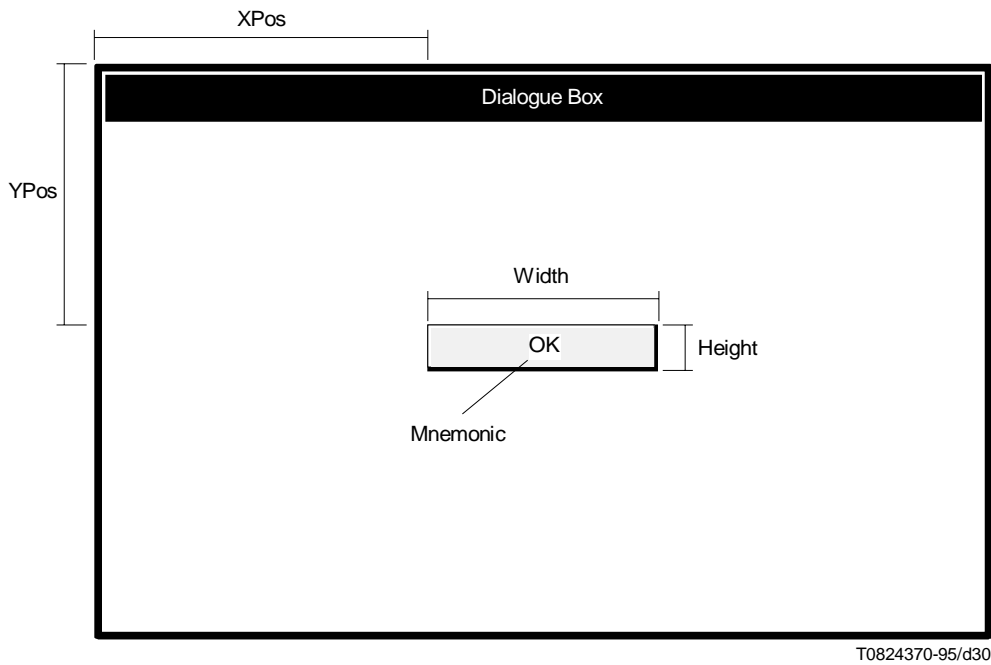


FIGURE 31/T.107
Push button

Attributes

- CIN: This attribute carries the Component Identification Number.
- XPos: This attribute carries the horizontal position of the element in NDC.
- YPos: This attribute carries the vertical position of the element in NDC.
- Width: This attribute carries the width of the component.
- Height: This attribute carries the height of the component.
- Closed: The element shall be in the closed state.
- NotAccessible: The element shall not be accessible.
- BIN: The BIN of the component.
- Text: The text content of the component.
- LocActAct: This attribute carries the code for the local action which is associated to the component and triggered by its activation.
- LocActVal: This attribute carries the code for the local action which is associated to the component and triggered by its validation.

7.6.1.8 The Text Input Field component

Description

The Text Input Field is composed of two items:

- the text label;
- the input area.

An attribute font can be associated to the text label. The input area has its starting location immediately after the text label. The type of the input data can be specified by the application. The following predefined types are available:

- any text character;
- alphabetic (A..Z, a..z, diacritical characters);
- numeric (0..9, +, -, comma, dot, space);
- alphanumeric (alphabetic, numeric).

An attribute of the Text Input Field specifies whether the user inputs are echoed or not. The VEMMI application can define one character to echo all user inputs.

A mnemonic key can be used to activate the Text Input Field (see 7.1.7).

Behaviour

If the space allocated for the input area in a one line Text Input Field is not sufficient to display all characters entered by the user, the input area shall offer horizontal scrolling facilities. If the space, allocated for the input area in a Text Input Field with more than one line, is not sufficient to display all characters entered by the user, the input area shall offer vertical scrolling facilities. In order to display all possible characters, it is recommended to give enough space for user inputs. If the maximum number of the input characters is reached this should be indicated to the user.

Constraints

The maximum length of the text label is limited to the first line of the Text Input Field component width. For one line input fields the label is placed in front of the input area and left aligned, while the input area is aligned to the right side. For multiline Text Input Fields it is placed in the top row and shall be left aligned. There is no space reserved for the text label if the attribute TextLabel is not present.

Interactive functionality

- activation;
- scrolling;
- local editing functionality of the input area (terminal dependent).

Visual aspect (see Figure 32)

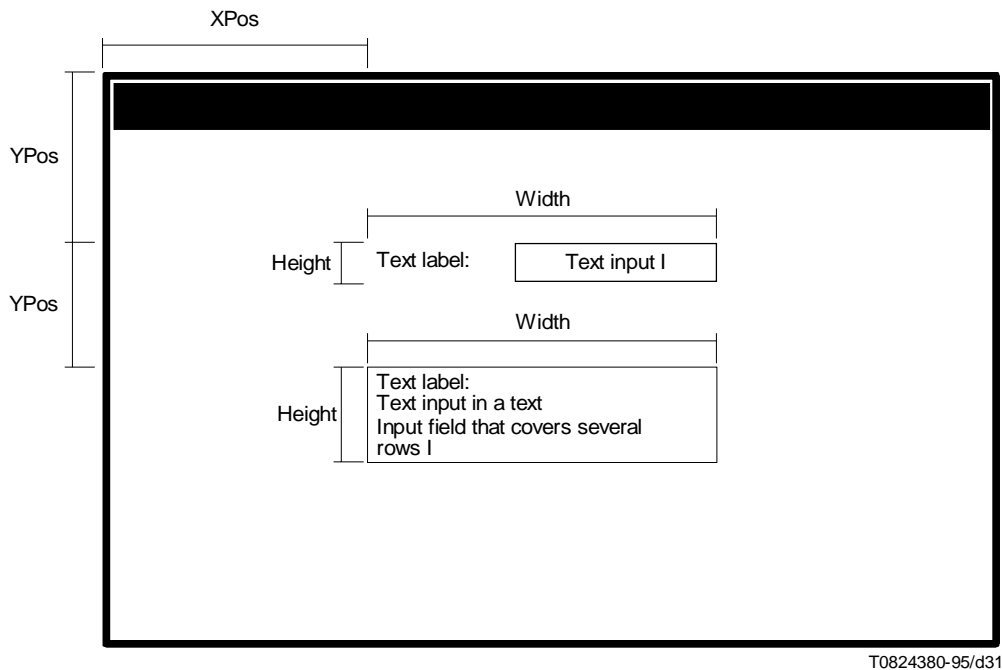


FIGURE 32/T.107
Text input field

Attributes

- **CIN:** This attribute carries the Component Identification Number.
- **XPos:** This attribute carries the horizontal position of the element in NDC.
- **YPos:** This attribute carries the vertical position of the element in NDC.
- **Width:** This attribute carries the width of the component.
- **Height:** This attribute carries the height of the component.
- **Closed:** The element shall be in the closed state.
- **NotAccessible:** The element shall not be accessible.
- **MaximTxt:** Local maximization can maximize the text of the component proportionally.
- **DefValue:** This attribute carries the default value for the component.
- **TextLabel:** This attribute carries the text label of the component.
- **LabelFont:** Specifies the FIN for the text label.
- **InputType:** This attribute specifies which type of user inputs shall be accepted by the component.
- **Echo:** This attribute specifies which type of echo shall be made on user inputs.
- **EchoChar:** This attribute carries the character that shall be displayed to echo user inputs.

- **MaxChar:** This attribute carries the maximum number of characters the user can enter in one line.
- **MaxLine:** This attribute carries the maximum number of lines the user can enter in a multiline input field.
- **Multiline:** Indicates a multiline input field.
- **InputTransformation:** The attribute specifies if the terminal shall convert the characters entered by the user to upper case characters, to lower case characters or if no conversion shall be done.
- **LocActAct:** This attribute carries the code for the local action which is associated to the component and triggered by its activation.
- **LocActVal:** This attribute carries the code for the local action which is associated to the component and triggered by its validation.

7.6.1.9 The Check Box component

Description

The Check Box component is composed of two items:

- the text label item;
- the check item.

A mnemonic key can be used to activate the component (see 7.1.7).

The check item shall support two different choices, visually identifiable, representing the two possible values marked or unmarked. The space allocated for the component shall include, in its dimensions, the space needed to represent this check item.

Behaviour

A Check Box acts like a switch. When switched by the user, its value changes from unmarked to marked or from marked to unmarked. The value of Check Boxes is independent of the value of any other components.

Interactive functionality

- activation;
- switch between the values marked and unmarked;
- validation.

Visual aspect (see Figure 33)

Attributes

- **CIN:** This attribute carries the Component Identification Number.
- **XPos:** This attribute carries the horizontal position of the element in NDC.
- **YPos:** This attribute carries the vertical position of the element in NDC.
- **Width:** This attribute carries the width of the component.
- **Height:** This attribute carries the height of the component.
- **Closed:** The element shall be in the closed state.
- **NotAccessible:** The element shall not be accessible.
- **TextLabel:** This attribute carries the text label of the component.
- **LabelFont:** FIN of the label.
- **DefMarked:** The default value of the element shall be marked.
- **LocActAct:** This attribute carries the code for the local action which is associated to the component and triggered by its activation.
- **LocActVal:** This attribute carries the code for the local action which is associated to the component and triggered by its validation.

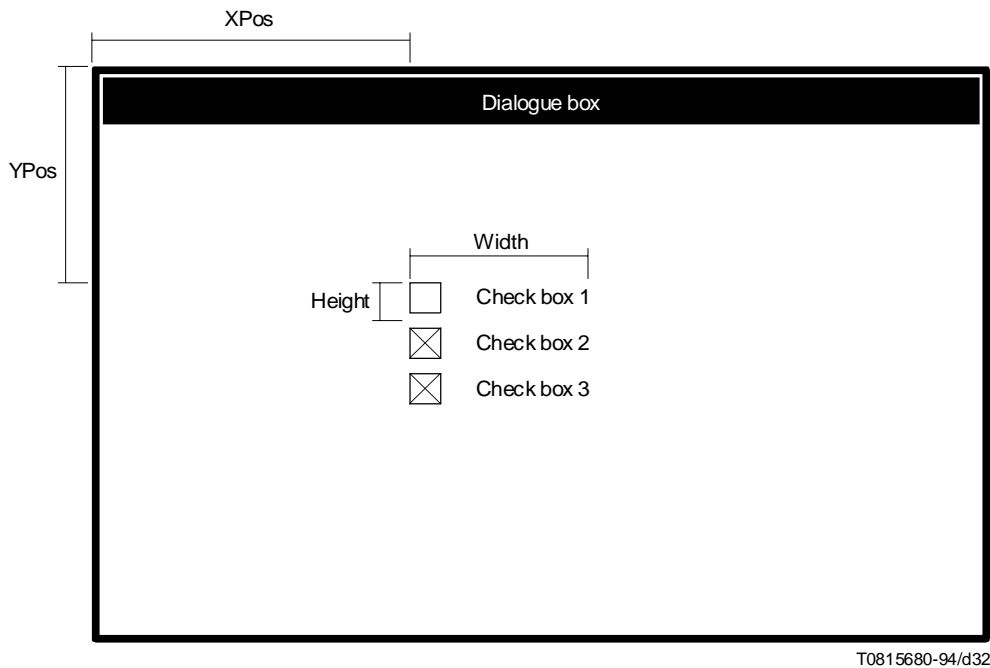


FIGURE 33/T.107
Check box

7.6.1.10 The Radio Button component

Description

The Radio Button component is composed of two items:

- the text label item;
- the check item.

A mnemonic key can be used to activate the component (see 7.1.7).

The check item shall support two different choices, visually identifiable, representing the two possible values marked and unmarked. The space allocated for the component shall include, in its dimensions, the space needed to represent this check item.

Radio Buttons are typically used in groups to provide a single-choice field.

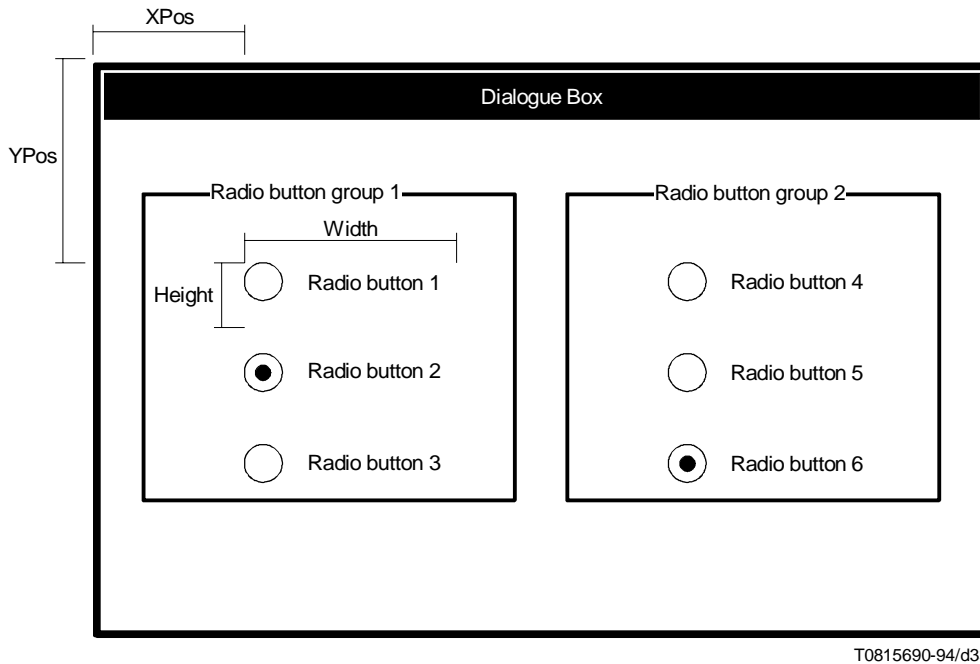
Behaviour

No more than one Radio Button shall be marked in a Radio Button group at a given time. The attempt to mark a second Radio Button shall cause the unmarking of the previously marked Radio Button, whatever its state (opened, closed, accessible, inaccessible) within the application may be.

Interactive functionality

- activation;
- switch between the values marked and unmarked;
- validation.

Visual aspect (see Figure 34)



T0815690-94/d33

FIGURE 34/T.107

Radio buttons with a frame and a text presentation area

Attributes

- CIN: This attribute carries the Component Identification Number.
- XPos: This attribute carries the horizontal position of the element in NDC.
- YPos: This attribute carries the vertical position of the element in NDC.
- Width: This attribute carries the width of the component.
- Height: This attribute carries the height of the component.
- Closed: The element shall be in the closed state.
- NotAccessible: The element shall not be accessible.
- TextLabel: This attribute carries the text label of the component.
- LabelFont: FIN of the label.
- Group: This attribute carries the identifier of the Radio Button group. Only one Radio Button in a group can be marked at one specific time.
- DefMarked: The default value of the element shall be marked.
- LocActAct: This attribute carries the code for the local action which is associated to the component and triggered by its activation.
- LocActVal: This attribute carries the code for the local action which is associated to the component and triggered by its validation.

7.6.1.11 The List Box component

Description

The List Box component is a rectangular area in which a list of text items is offered to the user for choice. If the total number of items exceeds the number of presented items, scrolling facilities shall be offered to the user through dedicated scrolling controls. When the entire list is visible the scrolling controls may be invisible. In this case, the space allocated for the component shall include in its dimensions the space needed to represent the scrolling facilities. The tools to provide the scrolling functionality are terminal dependent (vertical scrollbar, buttons, etc.).

Small icons may be added to the list items in order to visually emphasize their meaning. The referenced bitmap can be resized to fit to the local List Box representation.

The whole list is composed of list items. There shall be no CR + LF within the text content of a list item. If the attribute sorted is selected, the list items should be sorted in their alphabetical order. If sorted is not selected, the list shall be presented in the ascending order of the list item indices. However, the correspondence between the list items and the indices is always maintained.

Behaviour

When the List Box is active the terminal shall offer shifting facilities to the user. A List Box contains a list of items that the user can scroll through and select from. Single or multiple choice can be made by the user, depending on the list attributes.

There may be local actions associated to each List Box item. Additionally, the List Box may also have local actions associated. Upon activation and validation of a List Box item, the local actions associated to the list box item shall be executed. If there is no local action associated to the selected item, the local action associated to the List Box itself shall be executed. In this case, any OIN appearing in element specific commands of the List Box component is incremented by the value of the list index of the list box item.

A List Box with multiple choice cannot be validated.

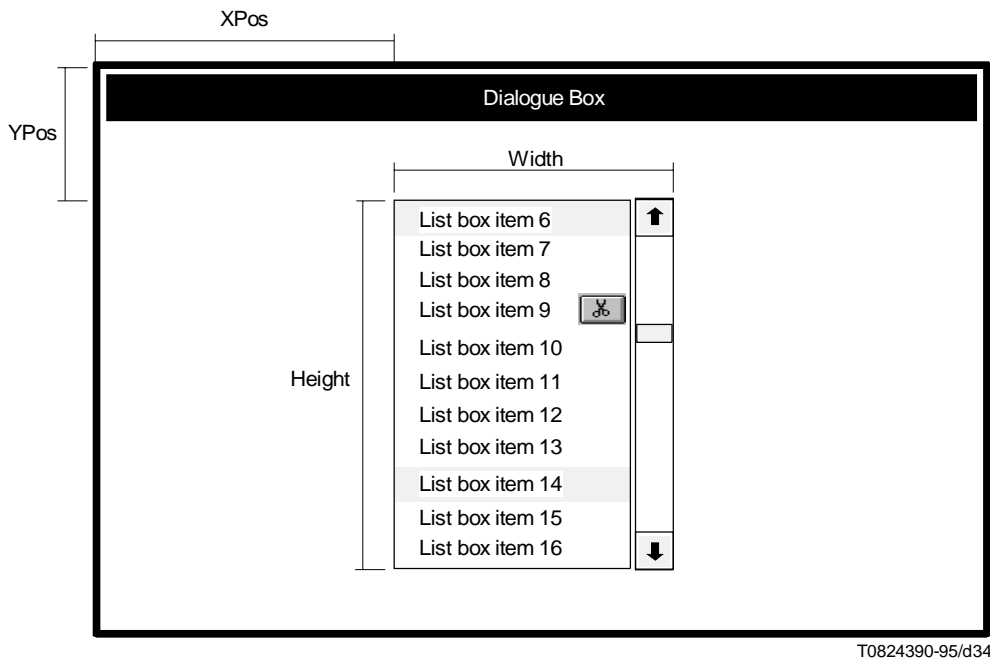
Interactive functionality

- activation;
- single or multiple choice;
- validation (single choice);
- scrolling;
- shifting.

Visual aspect (see Figure 35)

Attributes

- CIN: This attribute carries the Component Identification Number.
- XPos: This attribute carries the horizontal position of the element in NDC.
- YPos: This attribute carries the vertical position of the element in NDC.
- Width: This attribute carries the width of the component.
- Height: This attribute carries the height of the component.
- Closed: The element shall be in the closed state.
- NotAccessible: The element shall not be accessible.
- DefItem: This attribute carries the index of the list item that is selected by default when the component is opened for the first time.
- Sorted: The list items are presented sorted in alphabetical order.
- MultipleChoice: The element allows multiple choices.
- LocActAct: This attribute carries the code for the local action which is associated to the component and triggered by its activation.
- LocActVal: This attribute carries the code for the local action which is associated to the component and triggered by its validation.



T0824390-95/d34

FIGURE 35/T.107

List box

List item attributes

- **ListIndex:** This attribute carries the index of the next list item in the component description. It allows in a modify command the modification of a specific list item. The list indices shall be defined in an ascending order. The list indices may not be consecutive in order to allow easy updating of the list (inserting of list items). The terminal shall present a consecutive list to the user, sorted by the ascending value of the list indices corresponding to the list text. If in modify command a list index for a list item is used which is already existing, then the existing list item is replaced. If in a modify command a list index of a list item is used that is not existing in the terminal, the new list item shall be added. If in a modify command a list index is referenced which is already existing but in the modify command the corresponding attribute "ListText" is missing, the referenced list item on the terminal shall be deleted.
- **ListText:** This attribute carries the text content of a list item.
- **BIN:** The BIN of the component.
- **ItemActAct:** This attribute carries the code for the local action which is associated to the item and triggered by its activation.
- **ItemActVal:** This attribute carries the code for the local action which is associated to the item and triggered by its validation.

7.6.1.12 The Combination Box component

Description

The Combination Box is composed of two parts:

- a single choice List Box;
- a one line text field located at the top of the list.

If the total number of items exceeds the number of presentable items, scrolling facility shall be offered to the user through dedicated scrolling facilities. In this case, the space allocated for the component shall include in its dimensions the space needed to represent the scrolling facilities. The tools to provide the scrolling functionality are terminal dependent (vertical scrollbar, buttons, etc.).

Small icons may be added to the list items in order to visually emphasize their meaning. The referenced bitmap can be resized to fit to the local Combination Box representation.

The whole list is composed of list items. There shall be no CR + LF within the text content of a list item. If the attribute sorted is selected, the list items should be sorted in their alphabetical order. If sorted is not selected the list shall be presented in the ascending order of the list item indices. However, the correspondence between the list items and the indices is always maintained.

A variation of the Combination Box is a drop down Combination Box. It is composed of a Combination Box and a Push Button. Only the text field and the Push Button are displayed until the user validates the associated Push Button. The validation of the Push Button causes the display of the associated List Box.

Behaviour

When the List Box is active the terminal shall offer shifting facilities to the user. A Combination Box contains a list of items that the user can scroll through and select from to complete the text field. A parameter of the Combination Box specifies if the text field is editable or not. If the text field is editable, the user can type text directly into the text field. A parameter of the component specifies if the text entered by the user shall match one of the items contained in the list.

There may be local actions associated to each List Box item. Additionally, the Combination Box may also have local actions associated. Upon activation and validation of a List Box item, the local actions associated to the list box item shall be executed. If there is no local action associated to the selected item, the local action associated to the Combination Box itself shall be executed. In this case, any OIN appearing in element specific commands of the List Box component is incremented by the value of the list index of the listbox item.

If the Combination Box is designed to be a drop down Combination Box, the validation of the Push Button shall switch between the display and the hide of the List Box.

Interactive functionality

- activation;
- single selection;
- validation;
- scrolling;
- editing.

Visual aspect (see Figure 36)

Attributes

- CIN: This attribute carries the Component Identification Number.
- XPos: This attribute carries the horizontal position of the element in NDC.
- YPos: This attribute carries the vertical position of the element in NDC.
- Width: This attribute carries the width of the component.
- Height: This attribute carries the height of the component.
- Closed: The element shall be in the closed state.
- NotAccessible: The element shall not be accessible.

- DefItem: This attribute carries the index of the list item that is selected by default when the component is opened for the first time.
- NoEdit: The text field of the component shall not be editable.
- NoConsistency: This attribute is applied if “NoEdit” is not set. It indicates that the text entered by the user can be any text string, not necessarily one of the defined text items.
- MaxChar: This attribute carries the maximum number of characters the user can enter in the component. It shall only be present if the “EditableInput” Parameter is set true.
- NoDropDown: The element shall not be dropped down.
- NotSorted: The list items are presented not sorted in alphabetical order.
- LocActAct: This attribute carries the code for the local action which is associated to the component and triggered by its activation.
- LocActVal: This attribute carries the code for the local action which is associated to the component and triggered by its validation.

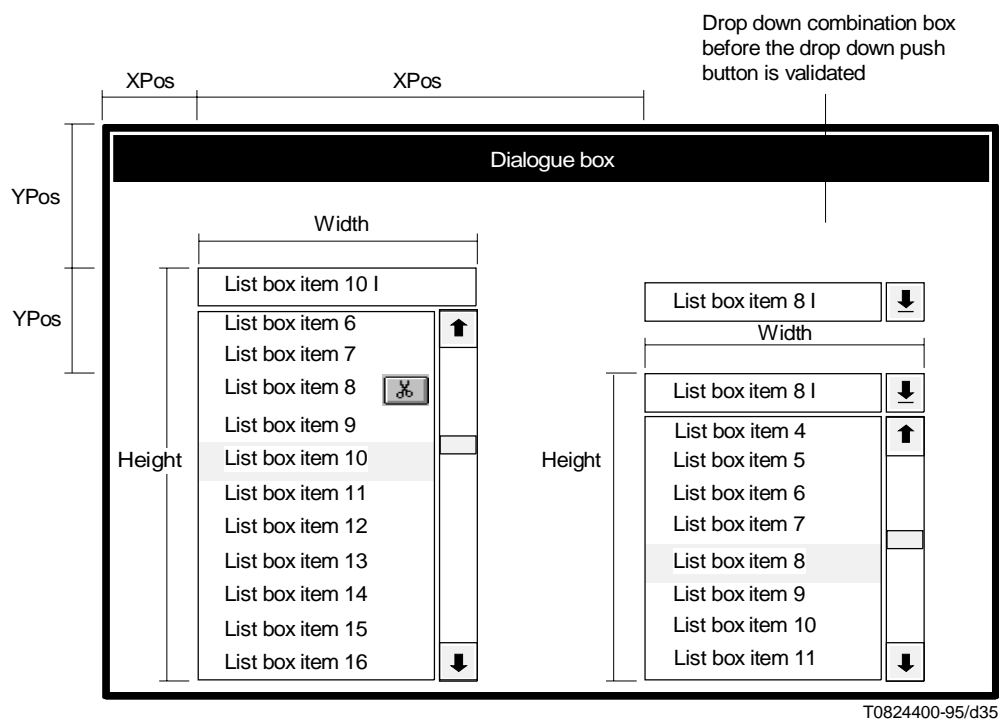


FIGURE 36/T.107
Combination box

List item attributes

- ListIndex: This attribute carries the index of the next list item in the component description. It allows in a modify command the modification of a specific list item. The list indices shall be defined in an ascending order. The list indices may not be consecutive in order to allow easy updating of the list (inserting of list items). The terminal shall present a consecutive list to the user, sorted by the ascending value of the list indices corresponding to the list text. If in modify command a list index for a list item is used which is already existing, then the existing list item is replaced. If in a modify command a list index of a list item is used that is not existing in the terminal, the new list item shall be added. If in a modify command a list index is referenced which is already existing but in the modify command the corresponding attribute "ListText" is missing, then the referenced list item on the terminal shall be deleted.
- ListText: This attribute carries the text content of a list item.
- BIN: The BIN of the component.
- ItemActAct: This attribute carries the code for the local action which is associated to the item and triggered by its activation.
- ItemActVal: This attribute carries the code for the local action which is associated to the item and triggered by its validation.

7.6.1.13 The Slider component

Description

The slider offers the selection of an analogue value by moving an adjustable marker on a slide bar between a minimum and a maximum value. The intervals are set by the application.

The Slider component consists of the following items:

- a) Slider (with the movable marker, the initial marker position shall correspond to the default value, set by application);
- b) Text Label (naming the slider);
- c) MinLabel (the lowest selectable value), optional;
- d) MaxLabel (the highest selectable value), optional;
- e) DirectInputField (optional), an input and output field which contains:
 - 1) the initial value when the Slider is activated;
 - 2) the values in text form corresponding to the slider movements;
 - 3) an Input field for user inputs of the slider position.

The Slider Component may be oriented in vertical and horizontal direction. All items shall fit in the rectangle determined by the component dimensions.

A local action which is triggered on the validation shall be performed when the user deactivates the component.

Behaviour

When the Slider Component is active the marker can be moved by striking the cursor keys or by mouse clicks. If the optional DirectInputField is present, the user is able to put in values directly by key strokes. The changed values shall be displayed immediately in the DirectInputField. The marker shall be moved onto a position which is relative to this value input. In the other case, when the marker is moved, the displayed value shall refer to this movement. The MinLabel and the MaxLabel (If they are present) are indicating the range, within a choice is available.

Interactive functionality

- local input functionality of the DirectInputField (application dependent);
- local positioning of the slider marker;
- validation by deactivation of the component;
- validation.

Visual aspect (see Figure 37)

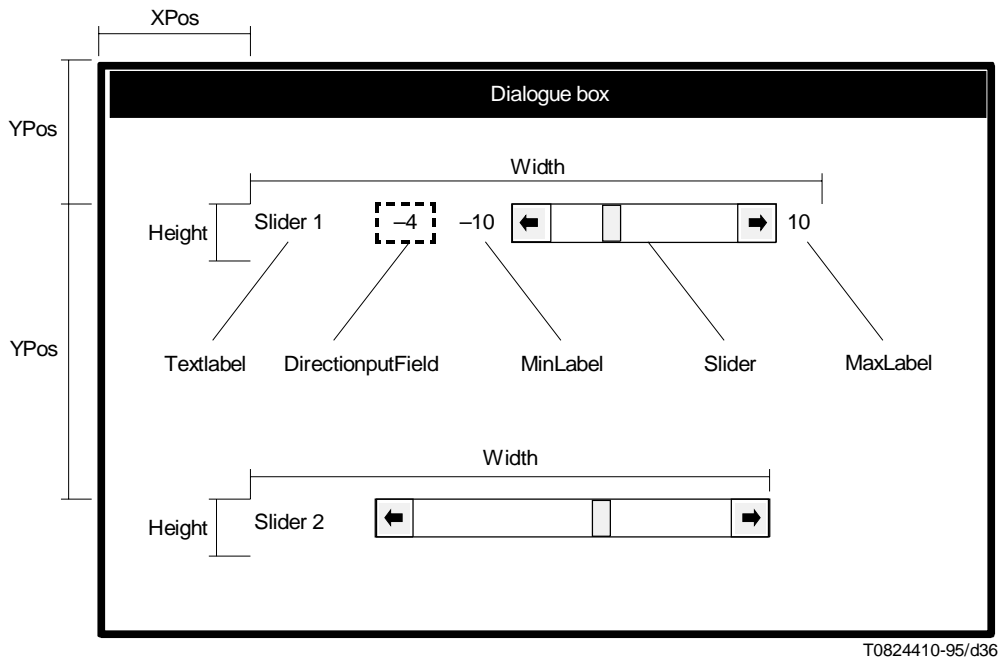


FIGURE 37/T.107

Slider component

Attributes

- CIN: This attribute carries the Component Identification Number.
- XPos: This attribute carries the horizontal position of the element in NDC.
- YPos: This attribute carries the vertical position of the element in NDC.
- Width: This attribute carries the width of the component.
- Height: This attribute carries the height of the component.
- Closed: The element shall be in the closed state.
- NotAccessible: The element shall not be accessible.
- Label: This attribute carries an alphanumeric text string.
- LabelFont: FIN of the label.

- **MinValue:** The lowest adjustable value.
- **MaxValue:** The highest adjustable value.
- **Increment:** The interval size between two adjustable values.
- **InitialValue:** The initial value, when the Slider Component is opened.
- **Negative:** This attribute can be associated with MinValue, MaxValue, InitialValue and indicates that the values are negative.
- **DirectIn:** This attribute indicates that MinValue and MaxValue should be displayed and the user's direct input should be allowed.
- **Vertical:** The element shall be presented vertically.
- **LocActAct:** Specifies the code for the local action which is associated to the component and triggered by its activation.
- **LocActVal:** Specifies the code for the local action which is associated to the component and triggered by its validation.

7.6.1.14 The Sensitive Area component

Description

A Sensitive Area component is an area, non materialized, in which the application permits a validation operation. The Sensitive Area component is intended to be used associated with a Graphic Output Area component of the Dialogue Box. For the use with text components, the in-text attributes for the definition of sensitive text are more appropriate. The sensitive area is a rectangle inside the dialogue box.

If the attribute locator is set, the terminal shall send upon validation not only its CIN to the VEMMI application, but also the coordinates of the cursor in NDC at the time of the validation with respect to the component origin.

Behaviour

When active a visible effect (e.g. thread, dot lines) can be implemented.

Constraints

A Sensitive Area shall not be covered by other components or items which are sensitive to user interaction, in order to avoid conflicts.

Interactive functionalities

- activation;
- validation.

Visual aspect (see Figure 38)

Attributes

- **CIN:** Specifies the Component Identification Number.
- **XPos:** Specifies the horizontal position sensitive area.
- **YPos:** Specifies the vertical position of the sensitive area.
- **Width:** Specifies the width of the sensitive area.
- **Height:** Specifies the height of the sensitive area.
- **Closed:** The element shall be in the closed state.
- **NotAccessible:** The element shall not be accessible.

- **Locator:** The position of the cursor in NDC at the time of the validation with respect to the component origin shall be sent with each report command.
- **LocActAct:** Specifies the local actions which are associated to the component and triggered by its activation.
- **LocActVal:** Specifies the local actions which are associated to the component and triggered by its validation. If the locator attribute is set, each report action shall also contain the coordinates of the cursor in NDC at the time of the validation with respect to the component origin.

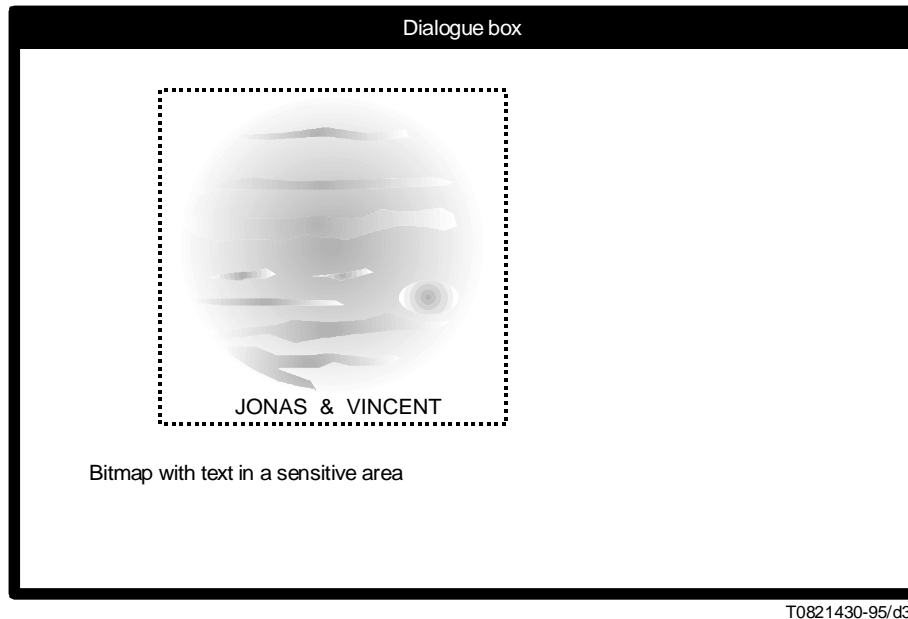


FIGURE 38/T.107
Bitmap, text and sensitive area

7.7 The Message Box

Description

The Message Box is a rectangular area in the DDA which contains text information.

Four specified types of message are defined:

- general message;
- information message;
- warning message;
- action message.

This information can be used to add an icon characterizing the type of the message to the presentation of the message box.

An attribute of the Message Box specifies if the terminal should perform a sound at the time the Message Box is opened.

The Message Box can be provided with a border area which should be equal to one character position. When a border is requested, a frame shall be drawn by the terminal. This frame drawing is terminal dependent.

Behaviour

An attribute of the Message Box specifies the lifetime of a Message Box. The following three values are available:

- destroy by any user interaction;
- close by any user interaction;
- destroy by the user validation of an implicitly defined button;
- close by the user validation of an implicitly defined button;
- no implicit lifetime is defined.

The destroying or closing of the Message Box shall not induce any report to the server.

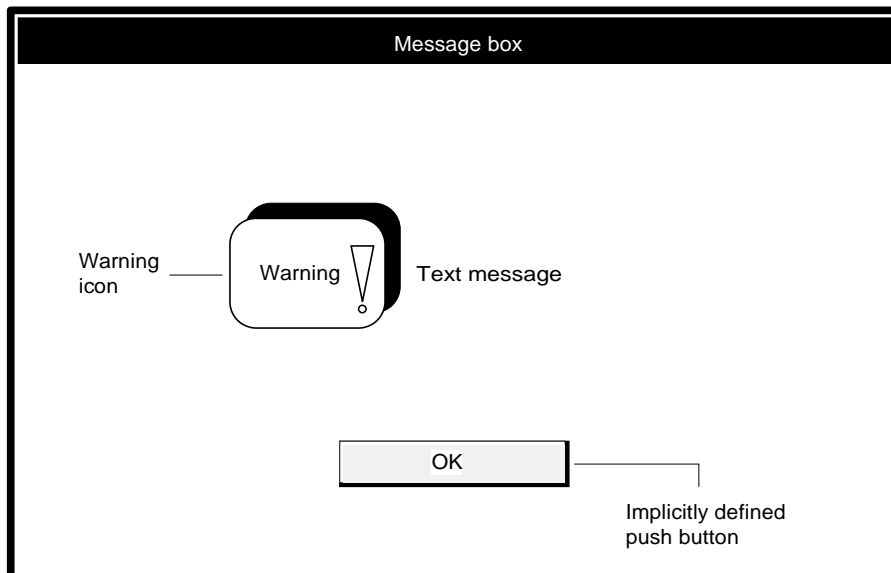
If no position is defined by the VEMMI application, the terminal shall centre the Message Box in the DDA.

If no dimensions are defined by the VEMMI application, the terminal shall calculate the appropriate size of the Message Box in order to have enough room to display the text message and the possible Push Button. The label and the size of the possible Push Button are terminal dependent.

Interactive functionality

- moving;
- validation.

Visual aspect (see Figure 39)



T0815790-94/d38

FIGURE 39/T.107
Message box with a warning message

Attributes

- XPos: Specifies the horizontal position of the message box.
- YPos: Specifies the vertical position of the message box.
- Width: Specifies the width of the message box.
- Height: Specifies the height of the message box.
- Closed: The element shall be in the closed state.
- MessageType: One of: general, information, warning, action.
- Modal: The element shall be modal.
- NoBorder: The element shall not have a border.
- AttributedText: Text of the message with in-text attributes.
- MaxTime: Time period in seconds.
- Lifetime: The events which lead to the destruction of the message box.
- NoSound: The terminal should not perform a sound when the message box is opened.

7.8 Operative object

With this object an application references a program which will be linked to the VEMMI application. This object type provides a method to extend the capabilities of an application during runtime. Via the service VEMMI_Create_Object the host application defines the operative object. The program is started with VEMMI_Open_Object. After termination the control is given back to the application.

The program itself has been downloaded via file transfer, is part of the operating system or its presence has been negotiated in another way. It is referenced with its filename. It is up to the host to ensure that the called program is started faultless, by providing the correct program parameters. An often used method to extend the functions of programs during runtime is the use of Dynamic Link Libraries, DLLs.

The program referenced with an operative object can be of the following types:

- standalone program;
- program with a filter interface.

For the purpose of operative objects, the services VEMMI_Create_Object VEMMI_Open_Object VEMMI_Destroy_Object are used. The program name and the program parameters are defined as attributes of the operative object.

Two parameters, program description and the source identification, can be used by the terminal to evaluate information to be presented to the user or only to be recorded. This precaution is necessary in order to help the user to detect programs which have been linked to the VEMMI application with a sabotage intent. Their effect on the terminal may not be harmless. The terminal can answer the host “The user refused the start of the program!”. On which basis such an answer is generated is application dependent, but at least the information is available on a structured basis. If the terminal cannot find the program, the host has to be informed with a specific error message.

Requirements for standalone programs

The program has no data exchange channel with the VEMMI local manager during the execution. Program parameters can be defined by the host application and supplied to the program when started. The operative object has no visual effect on the terminal and cannot be activated directly by the user, but the started program may use the local UI on an application specific basis. See Figure 40.

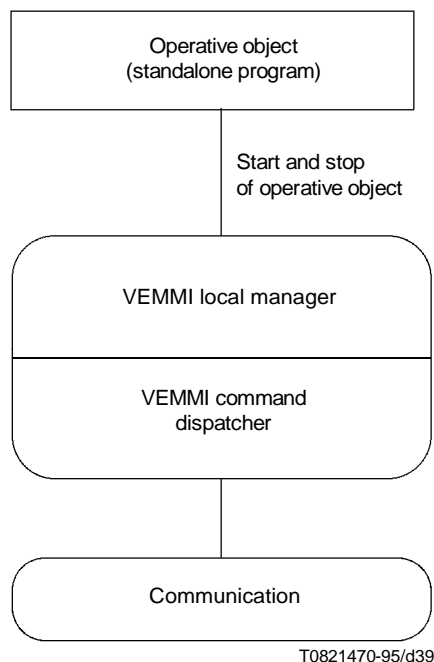


FIGURE 40/T.107
Standalone programs

The VEMMI command dispatcher is responsible for delivering the VEMMI commands between the VEMMI local manager and the communication part. All these interfaces are terminal and implementation dependent. They are used in this description to explain the integration of the two types of operative objects in the VEMMI terminal architecture.

Requirements for programs with filter interface

The program with filter interface has the possibility to access the VEMMI command dispatcher. The command dispatcher forwards the commands to the operative object, which decides to process them or forward them to the VEMMI local manager. This mechanism works also for the sending direction. The operative object has no visual effect on the terminal and cannot be activated directly by the user, but the started program may use the local UI on an application specific basis or through the VEMMI local manager. All these interfaces of such a program are terminal and implementation dependent and are not further described in this Recommendation. If an operative object receives a VEMMI_Close_Object with its own OIN from the host or it is terminated by the VEMMI local manager, it shall stop and the VEMMI local manager shall continue. All data exchange channels to the VEMMI local manager are disabled. See Figure 41.

Attributes

- Closed: The element shall be in the closed state.
- ProgName: The name of the program.
- ProgFilename: Filename of the program.
- ProgDescr: Description of the program.
- ProgAbout: Source identification of the program.
- ProgPar: List of parameters for the start of the program (optional).
- ProgType: Program type, standalone or with filter interface.

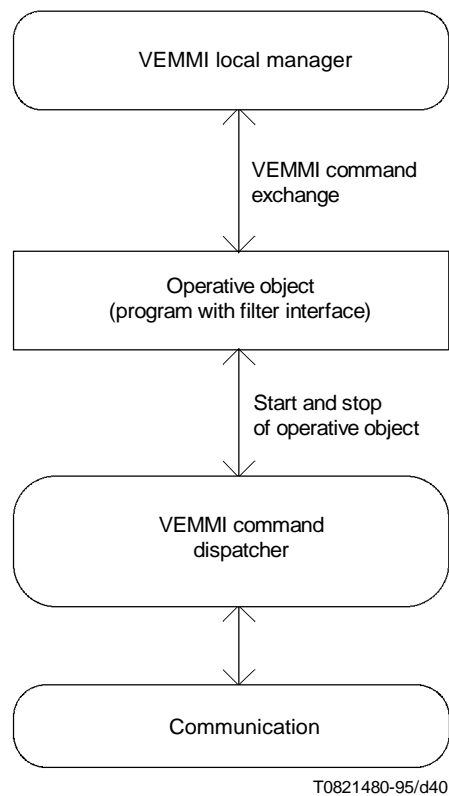


FIGURE 41/T.107
Programs with filter interfaces

7.9 Bitmap resource object

Description

A bitmap object contains either the bitmap definition itself or only a reference to a file with the bitmap definition.

Direct Bitmap Definition

The pixel matrix contains indices into the colour table or the RGB-components for each pixel. The attribute list is composed of two parts: the first is common to the two cases and the second is dependent from the colour definition. The first part is as follows:

- BmWidth: Width of the bitmap in pixel.
- BmHeight: Height of the bitmap in pixel.
- BmCompr: Compression type for the colour list. This value is currently not defined and only uncompressed colour lists can be specified (optional).

In the case of a colour index definition, the second part of the attribute list is as follows:

- BmBitsPerPixel: Number of bits per pixel. The value must be 1, 4 or 8 (optional, default value: 1).
- BmClrEntry: The colour table index on which all other colour indices from the index list are based (optional, default value: 0).

- **BmClrIdxList:** List of colour indices. This list contains $BmWidth * BmHeight$ colour indices. Depending on the number of bits per pixel, each colour index is in the range $0 .. 2^{BmBitsPerPixel}-1$. The colour indices are ordered row-by-row from left to right. The start point is the lower left corner of the bitmap. A colour index 0 means a colour table index equal to **BmClrEntry**. All other indices are relative to **BmClrEntry**.

In the case of a colour component definition, the second part of the parameter list is as follows:

- **BmBitsPerComp:** Number of bits per colour component. The value must be in the range 1 to 8 (optional, default value: 8).
- **BmClrCompList:** List of RGB triplets. This list contains $BmWidth * BmHeight$ triplets. The RGB-triplets are ordered row-by-row from left to right. The start point is the lower left corner of the bitmap.

File Bitmap Definition

Such a bitmap is defined from a picture file stored in the terminal. The file has been transmitted with a resource transfer service or by other means. A parameter indicates the file coding. If the file could not be converted in a suitable bitmap (upon **VEMMI_Create_Object**) the terminal shall send an error message to the host.

- **Filename:** Name of the file with the bitmap definition.
- **PictFileType:** Type of the picture file (one of JPEG, GIF, BMP).

The picture files conform to one of the following specifications:

- **JPEG:** Baseline system from ISO/IEC IS 10918 | Rec. ITU-T.81 “Digital compression and coding of continuous-tone still images”
- **GIF:** Graphics Interchange Format (sm) Version 89a. Compuserve Incorporated Columbus, Ohio, USA. Although the Version 89a is referenced a terminal has to support only the functions contained in the Version 87.
- **BMP:** Microsoft Windows Device-Independent Bitmap (DIB) with RLE4, RLE8 and 1, 4, 8 or 24 bits per pixel.

7.10 Videotex resource object

Description

A Videotex object contains either the Videotex content itself or only a reference to a file with the Videotex content.

Attributes

- **VTX:** VTX content.
- **Filename:** Name of the file with the VTX content.

7.11 Text resource object

Description

This object defines text content as a resource which can be referenced via the “Text Identification Number” (TIN). It contains either the text content itself or a reference to a file with the text content. In both cases the text content can contain in-text attributes, with references to font resource objects.

Attributes – Direct text resource definition

- **Text:** Text.
- **FIN:** Reference to a font resource object.

Attributes – File text resource definition

- Filename: Name of the file with the text content.
- TextFileType: One of:
 - text;
 - references to a font resource objects and text.

7.12 Font resource object

This object combines a set of text attributes in on font resource which can be referenced via the FIN.

Attributes

- FnFamily: The font family: SWISS, ROMAN or FIXFONT. (optional, default value: SWISS).
- FnHeight: Character height. (optional, default value: 10).
- FnBold: Character weight bold (optional).
- FnUnderline: Underlined (optional).
- FnItalic: Italic (optional).
- FnColour: Font colour: colour index. (optional, default: black).

7.13 Metacode object

The metacode object contains VEMMI commands. This object provides an easy way to avoid unnecessary dialogue steps with the host application. The command VEMMI_Open_Object starts the processing of the metacode object and after termination it is automatically in the closed state. The commands (the content of the metacode object) are processed in the same way as if they were received from the host application. For the content of the metacode object the following restrictions apply:

- the creation of a metacode object is not allowed;
- only host commands are allowed;
- a VEMMI_Open_Object command of another metacode object is not allowed (a VEMMI_Open_Blocking_Object is allowed).

Attributes

- VEMMICommands: VEMMI commands

7.14 VEMMI bitmap data type definition

See 7.9.

7.15 The VEMMI content encoding identification catalogue

Table 38 provides an overview of content encoding identifications defined in the VEMMI catalogue:

TABLE 38/T.107

VEMMI content encoding identification catalogue

For text data encoding:	
	T.51 String as defined in Annex A
	Recommendation T.52 [3]
	VEMMI high quality text
	ISO 8859-Series [13]. (Fully-formed accented char.)
	ISO 10646-1 [15] [multi-octet character set (Unicode)]
	Shift JIS Code (for Japanese characters)
For still picture data encoding:	
	Annex F/T.101 [4] (Photovideotext)
	ISO 10918-1 [16] (JPEG)
	VEMMI device independent bitmap
	Graphics Interchange Format (sm) Version 98a. CompuServe Incorporated Columbus, Ohio, USA
	Microsoft Windows Device-Independent Bitmap (DIB) with RLE4, RLE8 and 1, 4, 8 or 24 bits per pixel
For graphic data encoding:	
	Annex B/T.101 and Annex C/T.101 [4] (Videotex)
	ISO 8632 [12] (CGM)
For audio data encoding:	
	Annex E/T.101 [4]
	WAVE format
	MIDI format
For moving picture data encoding:	
	ISO 11172-2 [18] (MPEG Video)
	Recommendation H.261 [7] (videophone)
For audiovisual data encoding:	
	ISO 11172-1 [17] (MPEG System)
	Recommendation H.320 [8] (videophone)

8 Complete coded representation of the VEMMI

8.1 Introduction

This clause contains the syntax notation, the VEMMI header and the syntax of the commands, objects, components and local actions.

8.2 Notation used

The notation a/b with a, b = 0..15 denotes a byte value in hexadecimal.

The bits in a byte are noted b₈, b₇, ..., b₁. The bit b₈ is the high-order or most significant bit.

The syntax of the commands, objects, components and local actions is defined with a formal grammar using the following notation:

<symbol>	– nonterminal;
<SYMBOL>	– terminal;
<a/b>	– terminal, hexadecimal value: a, b = 0..15;
<symbol>*	– 0 or more occurrences;
<symbol>+	– 1 or more occurrences;
<symbol>o	– optional (0 or 1 occurrences);
<symbol-1> := <symbol-2>	– symbol-1 has the syntax of symbol-2;
<symbol-1> <symbol-2>	– symbol-1 or alternatively symbol-2;
<symbol-1 : symbol-2>	– symbol with the stated value;
{ comment }	– explanation of a symbol or production.

8.3 Overall switching of coding environment

ISO/IEC 9281 [14] describes a technique for identifying coding methods. The videotex VEMMI mode is one of the coding methods identified by ISO/IEC 9281 [14]. The diagram in Figure 42 gives an overview of the relationship between the videotex data syntaxes and ISO/IEC 9281 [14] coding environments.

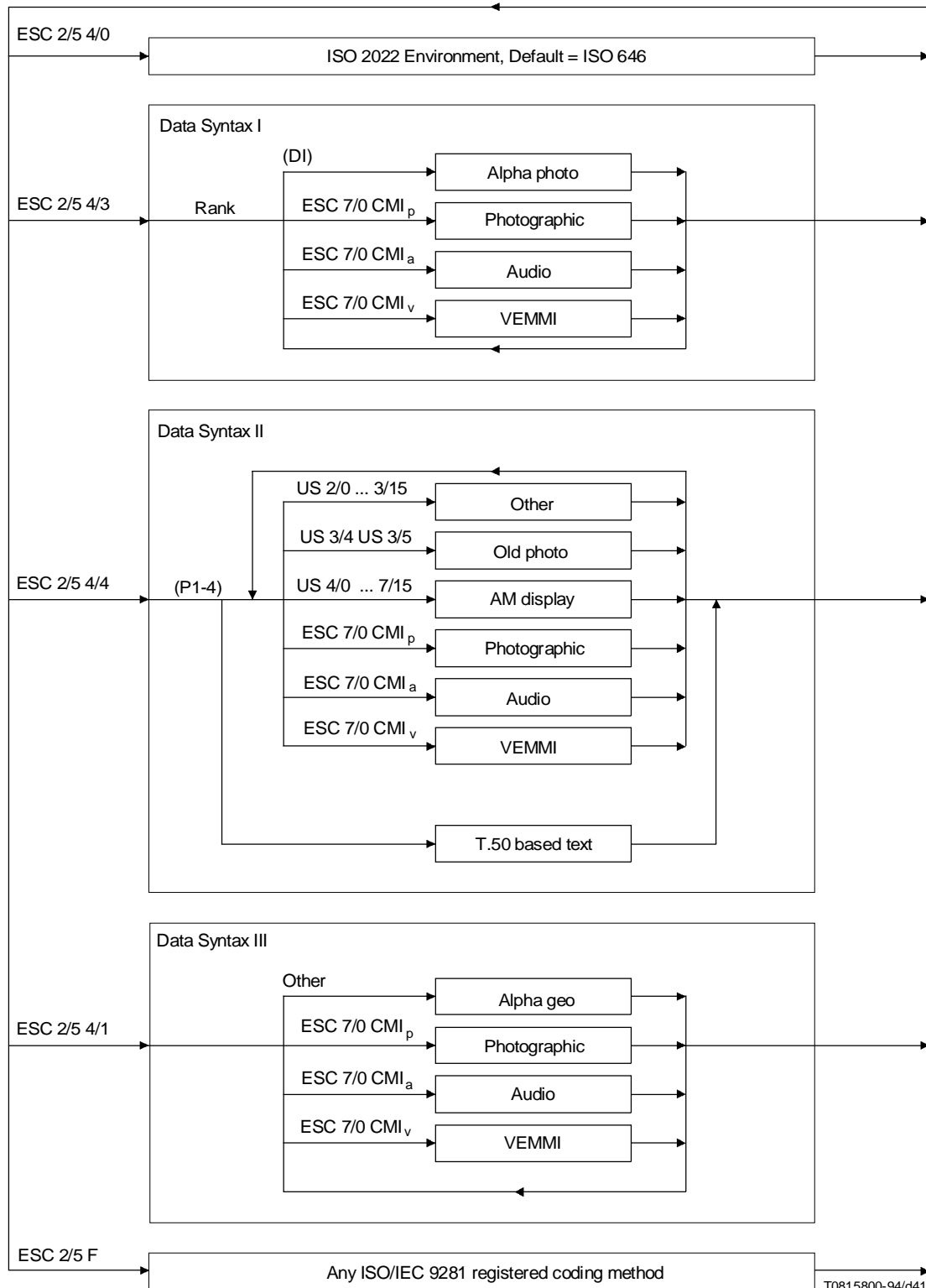
From an ISO 2022 [10] environment, a videotex data syntax can be explicitly entered using an ESC 2/5 F code. This is also the mechanism used for entering from an ISO 2022 [10] environment into an ISO/IEC 9281 [14] environment. The F code (“final byte”) is allocated and registered, according to ISO 2022 [10] by the registration authority ISO 2375 [11]. According to Appendix B/ISO 2375 [11], the videotex data syntaxes are regarded as “coding systems different to that of ISO 2022 [10]”. The F codes are 4/3 for CCITT data syntax I, 4/4 for CCITT data syntax II and 4/1 for CCITT data syntax III.

Since a videotex terminal usually begins operation, by default, in one of the data syntaxes, it shall not be mandatory to first send an ESC 2/5 F code (F is 4/1, 4/3 or 4/4). The diagram shows how these codes can be used to switch a videotex terminal supporting more than one data syntax from one data syntax to another.

8.3.1 Switching into the VEMMI mode

A videotex terminal operating within one of the data syntaxes (i.e. a coding system other than that described by ISO 2022 [10]) can enter the ISO/IEC 9281 [14] environment of the VEMMI mode according to their own rules. In the case of videotex for switching into the ISO/IEC 9281 [14] environment of the VEMMI mode, the Picture Code Delimiter (PCD) of the first Picture Element (PE) is used. The Coding Method Identifier (CMI) is used to distinguish between picture coding methods. In the case of videotex this shall be, for example, a distinction between audio, Photographic and VEMMI data.

NOTE – As ISO/IEC 9281 [14] was developed especially for the identification of picture coding in ISO 2022 [10] environments, the word “picture” is often used in the definitions, even when applicable to “audio”, for example. ISO has already accepted to make use of ISO/IEC 9281 [14] for non-pictorial information.



DI is data syntax I specific
P is a profile in data syntax II
F is a final code assigned by the ISO 2022 [10] registration authority
CMI_a is any CMI for videotex audio data
CMI_p is any CMI for videotex photographic data
CMI_v is any CMI for VEMMI data
Rank is data syntax I specific

FIGURE 42/T.107
Global switching mechanism

8.3.2 ISO/IEC 9281 [14] syntax structure

The high-level format of the syntax is as defined in ISO/IEC 9281 [14].

In the following description of the syntax 8-bit coding is assumed, thus the word “byte” is used with bit 8 set to zero. The coding described in ISO 9281 [14] is also valid in a 7-bit environment. In this case, the word “byte” shall be interpreted as meaning “7-bit byte” and the most significant bit, bit 8, shall not be used.

The structure of the coding is as follows:

```

PE      ::= PCE PDE;
PCE     ::= PCD CMI LI;
PCD     ::= 01/11 07/00;
CMI     ::= PM PI;
PM      ::= 02/05 (videotex VEMMI mode);
PI      ::= 04/00;
LI1)   ::= x111 1111<byte1><byte2>...<byten>;
<bytek> ::= x10D DDDD (k = n);
        | x11D DDDD (1 = <k<n).
    
```

x indicates do not care.

D indicates binary number 0 or 1.

Each piece of information, in this case encoded VEMMI data, is encoded as one or more Picture Entities (PEs). A PE (see Figure 43) consists of Picture Control Entity (PCE) which is followed by the actual data packed into a Picture Data Entity (PDE).

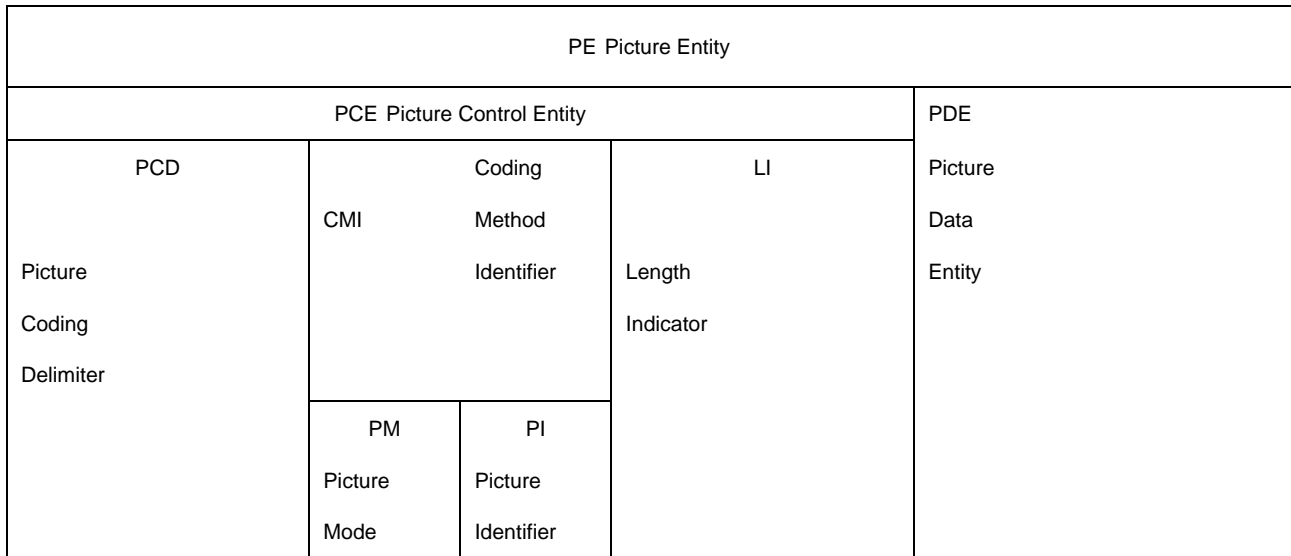


FIGURE 43/T.107

Structure of a Picture Entity

1) 5.2.7/ISO 9281 [14], should be consulted for the description of the Length Indicator.

NOTE 1 – In videotex VEMMI mode the size of a file containing an encoded VEMMI object can be rather large. One object may be transmitted in several PEs. The use of several PEs could facilitate the termination of the transmission of an object by the user.

The Picture Control Entity (PCE) consists of a Picture Coding Delimiter (PCD) and a Coding Method Identifier (CMI) followed by a Length Indicator (LI).

The Picture Coding Delimiter (PCD) is a fixed sequence of two octets: 01/11 07/00.

NOTE 2 – 01/11 is ESC.

The Coding Method Identifier (CMI) consists of a Picture Mode (PM) octet, followed by a Picture Identifier octet (PI). For VEMMI the PM is 02/05 as registered by ISO 9281 [14]. The PI octet has the value 04/00.

The length indicator (LI) specifies the number of bytes in the Picture Data Entity. Its encoding is described in the following (excerpt from ISO/IEC 9281 [14]):

b ₈	b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁
X	ONE	ZERO or ONE					

Bit b₈ of each byte shall be ignored. Bit b₇ of each byte shall be set to ONE. Bit b₆ of each byte shall be the Extension Flag. The LI value is specified in binary notation as an unsigned number using bits b₅ to b₁ with the weights 2⁴, 2³, 2², 2¹ and 2⁰, respectively. If the value of LI is less than, or equal to, 31 it shall be represented by one byte and the Extension Flag shall be set to ZERO. If the value of LI is larger than 31 it shall be represented by more than one byte. The most significant part of this value shall be recorded in the first byte. The Extension Flag shall be set to ONE in all bytes except the last where it shall be set to ZERO.

Values of LI:

- 1 byte: X111 1111
- 2 bytes: 0 ≤ LI ≤ 2⁵ – 1
- 3 bytes: 2⁵ ≤ LI ≤ 2¹⁰ – 1
- ⋮ ⋮ ⋮ ⋮ ⋮ ⋮ ⋮
- n bytes: 2⁵⁽ⁿ⁻¹⁾ ≤ LI ≤ 2⁵ⁿ – 1

Examples:

LI = 31

X	1	0	1	1	1	1	1
---	---	---	---	---	---	---	---

LI = 33

X	1	1	0	0	0	0	1
---	---	---	---	---	---	---	---

First byte

X	1	0	0	0	0	0	1
---	---	---	---	---	---	---	---

Second byte

8.4 VEMMI Command Syntax

Host command	Command syntax
VEMMI_Set_Translation_Mode	<translation mode opc> <translation mode>
VEMMI_Open	<open opc> <setup entity>
VEMMI_Close	<close opc>
VEMMI_Resume	<resume opc> <mode>
VEMMI_Suspend	<suspend opc>
VEMMI_Close_All	<close all opc>
VEMMI_Identify_Term_Cap	<term cap request opc> <cap request>
VEMMI_Set_Options	<set options opc> <text type list>o
VEMMI_Reset_Col_Table	<reset col opc>
VEMMI_User_Lock	<user lock opc>
VEMMI_User_Unlock	<user unlock opc>
VEMMI_Open_Object	<open object opc> <oin spec>
VEMMI_Close_Object	<close object opc> <oin spec>
VEMMI_Destroy_Object	<destroy object opc> <oin spec>
VEMMI_Obj_Access_Disable	<access disable opc> <oin spec>
VEMMI_Obj_Access_Enable	<access enable opc> <oin spec>
VEMMI_Create_Object	<create object opc> <entity create object>
VEMMI_Delete_Outdated_Objects	<delete outdated objects opc> <entity versionlist>+
VEMMI_Modify_Component (see also 9.6)	<modify comp opc> <entity modify comp>
VEMMI_Obj_Location_Change	<object loc change opc> <oin> <xloc> <yloc>
VEMMI_Load_Col_Table	<load coltable opc> <entity load coltable>
VEMMI_Open_Application	<open appl opc> <entity open appl>
VEMMI_Store_Objects	<store objects opc> <oin spec>
VEMMI_Erase_Objects	<erase objects opc> <oin spec>
VEMMI_Open_Blocking_Object	<open block object opc> <oin>
VEMMI_Resource_Transfer	<transfer opc> <transfer entity>
VEMMI_Store_Objects_Response	<store objects resp opc> <store result>
VEMMI_Object_Retransmission	<object retrans opc> <oin>
VEMMI_Identify_Term_Cap_Resp	<term cap response opc> <cap response>+
VEMMI_Open_Application_Response	<open appl resp opc> <entity open appl resp>
VEMMI_User_Data	<user data opc> <entity user data>
VEMMI_Error	<error opc> <entity errorreport>
VEMMI_Resource_Transfer_Abort	<transfer opc> <transfer id>

<translation mode>:=	<2/0> { no translation } <2/1> { 3 in 4 encoding } <2/2> { 7 shift encoding }
<setup entity>:=	<version> <mode>
<version>:=	<INTEGER> { Version }
<mode>:=	<native> <page based>
<native>:=	<INTEGER : 0> { VEMMI as described in this Recommendation }
<page based>:=	<INTEGER : 1> <basic page #> <input timeout>o
<basic page #>:=	<basic page # opc> <string>
<input timeout>:=	<input timeout opc> <INTEGER> { in seconds }
<cap request>:=	<data storage opc>o <local types opc>o <user language opc>o <system information opc>o
<cap response>:=	<version> <term cap list>
<term cap list>:=	<local storage info>o <data type info>o <user language>o <system information>o
<local storage info>:=	<local storage opc> <true> <false>
<data type info>:=	<data type opc> <text type> <still picture> <audio>o <graphic>o <moving picture>o <audio visual>o
<user language>:=	<user language opc>o <string> { list of alphabetical language codes encoded according to Annex B/ISO 639 (example: Dutch language: "NL") }
<system information>:=	<system information opc>o <string> <text type>:= <text type opc> <text type list>+ <end of list>
<still picture>:=	<still picture opc> <still picture list>+ <end of list>
<graphic>:=	not yet defined.
<audio>:=	<audio opc> <audio type list>+ <end of list>
<moving picture>:=	<moving picture opc> <moving picture list>+ <end of list>
<audio visual>:=	<audio visual opc> <audio visual list>+ <end of list>
<text type list>:=	<INTEGER : 0> { T.51String } <INTEGER : 1> { VEMMI high quality text } <INTEGER : 2> { ISO 8859-1 } <INTEGER : 3> { ISO 10646-1 } <INTEGER : 4> { Shift JIS code } <INTEGER : 5> { ISO 8859-2 } <INTEGER : 6> { ISO 8859-3 } <INTEGER : 7> { ISO 8859-4 } <INTEGER : 8> { ISO 8859-5 } <INTEGER : 9> { ISO 8859-6 } <INTEGER : 10> { ISO 8859-7 } <INTEGER : 11> { ISO 8859-8 } <INTEGER : 12> { ISO 8859-9 } <INTEGER : 13> { ISO 8859-10 }

<still picture list>:= <INTEGER : 0> | { *Annex F/T.101* }
 <INTEGER : 1> | { *JPEG* }
 <INTEGER : 2> | { *VEMMI DIB* }
 <INTEGER : 3> | { *GIF* }
 <INTEGER : 4> | { *MS DIB* }

<audio type list>:= <INTEGER : 0> | { *Annex E/T.101* }
 <INTEGER : 1> | { *WAVE* }
 <INTEGER : 2> | { *MIDI* }

<moving picture list>:= <INTEGER : 0> | { *ISO 11172-2* }
 <INTEGER : 1> | { *Recommendation H.261* }

<audio visual list>:= <INTEGER : 0> | { *ISO 11172-1* }
 <INTEGER : 1> | { *Recommendation H.320* }

<entity **create object**>:= <oin> <template>o <autostore>o <object>

<object>:= <display object> | <resource object> | <metacode object>

<entity **versionlist**>:= <timestamp> <oin spec>

<entity **modify comp**>:= <oin> <display object>

<entity **load coltable**>:= <colour entry>o <col rgb list>

<colour entry>:= <colour entry opc> <INTEGER>

<col rgb list>:= <list spec>

<entity **open appl**>:= <applid> <appl add data>o <timestamp> <basic page #>o <input timeout>o

<applid>:= <applid opc> <string>

<appl add data>:= <appl add data opc> <string>

<entity **open appl resp**>:= <result>

<entity **user data**>:= <oin> <component data>+

<component data>:= <cin report> | { *for menu choices, push buttons, pop-up menu, sensitive area, sensitive text* }
 <text report> | { *for text input fields* }
 <list string report> | { *for list box, combo box* }
 <boolean report> | { *for check box, radio button* }
 <slider report> | { *for slider* }
 <locator report> | { *for sensitive area with locator attribute set* }

<cin report>:= <INTEGER : 0> <cin>

<text report>:= <INTEGER : 1> <cin> <string>

<list string report>:= <INTEGER : 2> <cin> <string>

<boolean report>:= <INTEGER : 3> <cin> <>true> | <>false>

<slider report>:= <INTEGER : 4> <cin> <slider value> <negative>o

<locator report>:= <INTEGER : 5> <cin> <xloc> <yloc>

<entity **errorreport**>:= <error type> <error oin>o <error cin>o <error com code>o

<error type>:= <INTEGER : 0> | { *general error* }
 <INTEGER : 1> | { *unknown command* }
 <INTEGER : 2> | { *erroneous command* }
 <INTEGER : 3> | { *object syntax error* }
 <INTEGER : 4> | { *unexpected command* }
 <INTEGER : 5> | { *out of memory* }
 <INTEGER : 6> | { *cannot process audio objects* }
 <INTEGER : 7> | { *cannot process video objects* }
 <INTEGER : 8> | { *invalid colour index* }
 <INTEGER : 9> | { *file not found* }
 <INTEGER : 10> | { *conversion to bitmap failed* }
 <INTEGER : 11> | { *cannot process direct col. definition* }
 <INTEGER : 12> | { *operative object has been rejected by the user* }
 <INTEGER : 13> | { *out of local storage space* }
 <INTEGER : 14> | { *a closed object has been destroyed* }
 <INTEGER : 15> | { *service not supported* }

<error oin>:= <error oin opc> <INTEGER>

<error cin>:= <error cin opc> <INTEGER>

<error com code>:= <error com code opc> <a/b> { *a = 2..4, b = 0..15* }

<transfer entity>:= <data transfer> | <transfer abort>

<data transfer>:= <INTEGER : 0> <transfer id> <current block number> <current block data length>
 <block>

<transfer id>:= <INTEGER> { *unique transfer identifier for each file* }

<current block number>:= <INTEGER> { *from one to the total number of blocks* }

<current block data length>:= <INTEGER>

<block>:= <file&block> | <current block>

<file&block>:= <filespec> <current block>

<filespec>:= <filename> <creation date> <file length> <total number of blocks>

<creation date>:= <string> { *in the format YYYYMMDDHHMMSS* }

<file length>:= <INTEGER>

<total number of blocks>:= <INTEGER>

<transfer abort>:= <INTEGER : 1> <transfer id>

<current block>:= <current block opc> <current block data>

<current block data>:= <a/b>+ { *a, b = 0..15* }

8.5 Objects, components

<display object>:= <application bar> | <button bar> | <pop-up menu> |
 <dialogue box> | <operative object> | <sound object> |
 <video object> | <message box>

<application bar>:=	<appl bar opc> <application bar body> <menu item>+
<application bar body>:=	<xpos>o <ypos>o <height>o <width>o <closed>o <notaccessible>o <defactive>o <vertical>o
<menu item>:=	<bar menu choice> <pull down menu item>*
<bar menu choice>:=	<bar menu choice opc> <menu choice>
<pull down menu item>:=	<pull down menu choice> <cas menu choice>
<pull down menu choice>:=	<pull down menu choice opc> <menu choice> <separated>o
<cas menu choice>:=	<cas menu choice opc> <menu choice> <separated>o
<menu choice>:=	<cin> <height>o <width>o <notaccessible>o <text> <locactact>o <locactval>o
<button bar>:=	<button bar opc> <button bar body> <push button comp>+
<button bar body>:=	<xpos> <ypos> <width>o <height>o <closed>o <notaccessible>o <vertical>o <defactive>o <modal>o
<push button comp>:=	<push button comp opc> <cin> <height>o <width>o <closed>o <notaccessible>o <button> <locactact>o <locactval>o
<pop-up menu>:=	<pop-up menu opc> <pop-up menu body> <pop-up menu choice>+
<pop-up menu body>:=	<xpos> <ypos> <width>o <closed>o <notaccessible>o <title>o <title font>o <defactive>o <modal>o
<pop-up menu choice>:=	<pop-up menu choice opc> <menu choice> <separated>o <cas menu choice>*
<dialogue box>:=	<dialogue box opc> <dialog box body> <separator>* <frame>* <graphic output area>* <text area>* <text input field>* <box push button>* <check box>* <radio button>* <list box>* <combo box>* <slider>* <sensitive area>* <text component>* <sensitive text>*
<dialog box body>:=	<xpos>o <ypos>o <width>o <height>o <closed>o <notaccessible>o <no border>o <title>o <defactive>o <modal>o <maximizable>o <background>o <store ini values>o
<background>:=	<colour> <bitmapped>
<bitmapped>:=	<bin reference> <bitmap disptype>o <colour>o
<separator>:=	<separator opc> <cin> <xpos>o <ypos>o <width>o <height>o <closed>o <vertical>o <colour>o
<frame>:=	<frame opc> <cin> <xpos>o <ypos>o <width>o <height>o <closed>o <colour>o
<graphic output area>:=	<graphic output area opc> <cin> <xpos>o <ypos>o <width>o <height>o <closed>o <graphic type>
<graphic type>:=	<bitmap> <videotex>
<bitmap>:=	<bitmap disp type>o <bin reference>
<videotex>:=	<data syntax type> <direct videotex content> <vin reference>
<data syntax type>:=	<INTEGER : 0> { Annex B/T.101 } <INTEGER : 1> { Annex C/T.101 } <INTEGER : 2> { Annex D/T.101 }
<direct videotex content>:=	<videotex content opc> <nr of vtx bytes> <vtx bytes>+
<nr of vtx bytes>:=	<INTEGER> { number of Videotex bytes }

<text area>:=	<text area opc> <cin> <xpos>o <ypos>o <width>o <height>o <closed>o <initial font>o <maxim text>o <no scrolling tools>o <no format>o <no border>o <autoscroll>o <text area content>o
<text area content>:=	<cin reference> { <i>cin of a text component</i> } <in-text>+
<text component>:=	<text component opc> <cin> <prev text> <current text> <next text>o
<prev text>:=	<prev text opc> <cin reference> { <i>cin of a text component</i> }
<next text>:=	<next text opc> <cin reference> { <i>cin of a text component</i> }
<current text>:=	<current text opc> <tin reference> { <i>tin of text resource object</i> } <in-text>+
<in-text>:=	<text> <in-text string>
<in-text string>:=	<fin reference> { <i>fin of a font resource object</i> } <sensitive text reference>
<sensitive text reference>:=	<cin reference> <text> { <i>cin of a sensitive text component</i> }
<sensitive text>:=	<sensitive text opc> <cin> <notaccessible>o <locact>o <locactval>o
<text input field>:=	<text input field opc> <cin> <xpos>o <ypos>o <width>o <height>o <closed>o <notaccessible>o <maxim text>o <default text>o <label>o <label font>o <input type>o <echo type>o <echo char>o <max char>o <max line>o <multiline>o <input transform>o <locact>o <locactval>o
<box push button>:=	<box push button opc> <cin> <xpos>o <ypos>o <width>o <height>o <closed>o <notaccessible>o <button> <locact>o <locactval>o
<check box>:=	<check box opc> <cin> <xpos>o <ypos>o <width>o <height>o <closed>o <notaccessible>o <label>o <label font>o <marked>o <locact>o <locactval>o
<radio button>:=	<radio button opc> <cin> <xpos>o <ypos>o <width>o <height>o <closed>o <notaccessible>o <label>o <label font>o <group>o <marked>o <locact>o <locactval>o
<list box>:=	<list box opc> <cin> <xpos>o <ypos>o <width>o <height>o <closed>o <notaccessible>o <default item>o <sorted>o <multiple choice>o <locact>o <locactval>o <list text unit>*
<combo box>:=	<combo box opc> <cin> <xpos>o <ypos>o <width>o <height>o <closed>o <notaccessible>o <default item>o <sorted>o <no drop>o <no consistency>o <no edit>o <locact>o <locactval>o <list text unit>*
<list text unit>:=	<list index>o <text> <icon reference>o <item locact>o <item locval>o
<icon reference>:=	<bin reference>
<slider>:=	<slider opc> <cin> <xpos>o <ypos>o <height>o <width>o <closed>o <notaccessible>o <label>o <label font>o <negative>o <min value> <negative>o <max value> <increment>o <initial info>o <direct in>o <vertical>o <locact>o <locactval>o
<initial info>:=	<negative>o <initial value>
<sensitive area>:=	<sensitive area opc> <cin> <xpos>o <ypos>o <width>o <height>o <closed>o <notaccessible>o <locator>o <locact>o <locactval>o
<operative object>:=	<operative object opc> <closed>o <program name> <program filename> <program description> <program about> <program parameter>o <program type>
<sound object>:=	<sound object opc> <closed>o <filename> <sound format>
<sound format>:=	<INTEGER : 0> { <i>wave format</i> } <INTEGER : 1> { <i>MIDI format</i> }

<video object>:= <video object opc> <closed>o <filename> <moving picture list>o
 <message box>:= <message box opc> <xpos>o <ypos>o <width>o <height>o <closed>o <no border>o <title>o <modal>o <message type>o <max time>o <lifetime>o <no sound>o <attributed text>+
 <message type>:= <message type opc> <message>
 <message>:= <INTEGER : 0> | { *general message* }
 <INTEGER : 1> | { *information message* }
 <INTEGER : 2> | { *warning message* }
 <INTEGER : 3> | { *action message* }
 <lifetime>:= <lifetime opc> <destroy event>
 <destroy event>:= <INTEGER : 0> | { *destroy by any user action* }
 <INTEGER : 1> | { *destroy by validation of implicit defined button* }
 <INTEGER : 2> | { *destroy by a command from the host* }
 <INTEGER : 3> | { *close by validation of implicit defined button* }
 <INTEGER : 4> | { *close by any user interaction* }
 <INTEGER : 5> | { *no implicit lifetime defined* }
 <max time>:= <max time opc> <INTEGER> { *time period in seconds* }

<resource object>:= <bitmap object> | | <text object> |
 <videotex object>

<bitmap object>:= <bitmap object opc> <bitmap data> | <bitmap file>
 <bitmap data>:= <INTEGER : 0> <bmwidth> <bmheight> <bmcompr>o <pixel definition>
 <bmwidth>:= <INTEGER>
 <bmheight>:= <INTEGER>
 <bmcompr>:= <bmcompr opc> <INTEGER : 0> { *no compression* }
 <pixel definition>:= <index def> | <rgb def>
 <index def>:= <index def opc> <bits per pixel>o <colour entry>o <colour list>
 <bits per pixel>:= <bits per unit>
 <colour entry>:= <colour entry opc> <INTEGER>
 <colour list>:= <list spec> { *bmwidth * bmheight integers* }
 <rgb def>:= <rgb def opc> <bits per component>o <component list>
 <bits per component>:= <bits per unit>
 <component list>:= <list spec> { *3 * bmwidth * bmheight integers* }
 <bits per unit>:= <bits per unit opc> <INTEGER>
 <bitmap file>:= <INTEGER : 1> <filename> <pict file type>
 <pict file type>:= <INTEGER : 0> | { *JPEG* }
 <INTEGER : 1> | { *GIF* }
 <INTEGER : 2> | { *BMP* }
 := <fn family>o <fn height>o <fn bold>o <fn underline>o <fn italic>o <fn colour>o
 <fn family>:= <fn family opc> <family>

<family>:= <INTEGER : 0> | { *SWISS* }
 <INTEGER : 1> | { *ROMAN* }
 <INTEGER : 2> { *FIXFONT* }

<fn height>:= <fn height opc> <INTEGER>

<fn bold>:= <fn bold opc>

<fn underline>:= <fn underline opc>

<fn italic>:= <fn italic opc>

<fn colour>:= <fn colour opc> <INTEGER>

<text object>:= <text object opc> <in-text>+ | <text file>

<text file>:= <file name> <text file type>

<text file type>:= <INTEGER : 0> | { *"in-text" format* }
 <INTEGER : 1> { *plain text, incl. CR, LF* }

<videotex object>:= <videotex object opc> <vtx file>

<videotex file>:= <filename> <data syntax type>

<metacode object>:= <metacode object opc> <command>+ <command end>

8.6 Local actions

<locactact>:= <locactact opc> <local action>+

<locactval>:= <locactval opc> <local action>+

<local action>:= <report command> | <general command> | <specific command>

<report command>:= <loc command opc> <report type>

<report type>:= <INTEGER : 0> | { *report OIN, CIN* }
 <INTEGER : 1> | { *report current value* }
 <INTEGER : 2> | { *report all values* }
 <INTEGER : 3> { *report all changed values* }

<general command>:= <loc command opc> <general command type>

<general command type>:= <INTEGER : 50> | { *user lock* }
 <INTEGER : 51> { *set initial values and states in the parent object* }

<specific command>:= <loc command opc> <specific command type> <oin spec> | <cin spec>

<specific command type>:= <INTEGER : 100> | { *open components of the parent object* }
 <INTEGER : 101> | { *close components of the parent object* }
 <INTEGER : 102> | { *open objects* }
 <INTEGER : 103> | { *close objects* }
 <INTEGER : 104> | { *change components of the parent object to inaccessible* }
 <INTEGER : 105> | { *change components of the parent object to accessible* }
 <INTEGER : 106> | { *destroy objects* }
 <INTEGER : 107> { *open blocking object (only one)* }

9 Encoding

9.1 Command structure

The PDE of one Picture Entity carries one or more VEMMI commands. The last VEMMI command of one PE can be split on more than one PE. This is indicated with the first byte of the PDE, called “more data indicator”. The syntax of the PDE is as follows:

<PDE>:= <mdi> <command>+

<mdi>:= <2/14> | { the last command is only partly contained in this PE, it continues in the next PE; “MORE” }
 <2/15> { the last command is completely contained in the PE; “LAST” }

NOTE – Splitting commands over more than one PE is not reflected in the grammar specification.

One or several VEMMI commands are preceded by the VEMMI-Header. This header is used to identify VEMMI data in the data stream of the standard application. Such a sequence of VEMMI commands is called the Data Entity. The last command of a data entity can be split across more than one data entity. This is indicated by the MDI (More Data Indicator). Examples:

Three commands contained in one data entity:

VEMMI Header	MDI (last)	Data Entity		
		Command 1	Command 2	Command 3

Three commands in two data entities, one is split:

VEMMI Header	MDI (more)	Data Entity		VEMMI Header	MDI (last)	Data Entity	
		Command 1	Command 2 (first part)			Command 2 (last part)	Command 3

9.2 Object, component and attribute structure

Object structure

The objects have one of the following structures:

- a) <object code> <object body> <component>*
- b) <object code> <attribute>+
- c) <object code> <command>+

Component structure

<component code> <cin> <attribute>*

Attribute structure

The attributes have one of the following structures:

- a) <attribute code> { the attribute is defined only by the code };
- b) <attribute code, value> { the attribute is defined by the code and the value };
- c) <attribute> { the attribute is defined by another attribute };
- d) <attribute code, attribute> { the attribute is defined by the code and another attribute }.

9.3 Terminal symbols encoding

9.3.1 Opcodes

Opcodes are used to identify the parts of the VEMMI commands. They are used to encode:

- command codes;
- object codes;
- component codes;
- the MDI code;
- attribute codes.

The terminal symbol notation is: <a/b> with a, b = 0 .. 15 (hexadecimal notation).

9.3.2 Integers

This format is used to encode non negative values with no fractional part. An integer consists of a byte sequence of one or more bytes.

Coding rules:

- a) bit 8 set to 1 means no extension, it is the last byte of the sequence;
- b) bit 8 set to 0 and bit 7 set to 1 indicates an extension, it is not the last byte;
- c) the encoded value is specified:
 - 1) for one byte integers by bits 7 to 1;
 - 2) for multi-byte integers by bits 6 to 1 from all bytes except the last byte and bits 7 to 1 from the last byte;
- d) the most significant part of the value is coded in the first byte;
- e) the least significant part of the value is coded in the last byte.

Symbols using integers:

<INTEGER>

<false>:= <INTEGER : 0>

<true>:= <INTEGER : 1>

Example: Coded representation of the number 37.

b ₈	b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁
1	0	1	0	0	1	0	1
No extension	Integer value bits						

Example: Coded representation of the number 1741.

Byte 1	b ₈	b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁
	0	1	0	0	1	1	0	1
	Extension			Most significant bits				

Byte 2	b ₈	b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁
	1	1	0	0	1	1	0	1
	No extension	Least significant bits						

9.3.3 Enumerated

This datatype denotes a value from a set of standardized values.

They are encoded as integers with the notation: <INTEGER : n> with n as a decimal value.

9.3.4 Strings

This datatype denotes a sequence of characters with the following structure:

<string>:=	<latin string> <non latin string> <unicode string>
<latin string>:=	<character>+ <end of string>
<character>:=	<a/b> <c/d> <CR> <LF> <ESC> { a = 2..7, b = 0..15, c = 10, 15 characters coded according to ISO 8859 or T.51String }
<end of string>:=	<9/12> { ST (string terminator) from ISO 6429 }
<CR>:=	<0/13>
<LF>:=	<0/10>
<non latin string>:=	<shift JIS string> <T.52 string>
<shift JIS string>:=	<a/b> <c/d> <e/f> <CR> <LF> { a = 2..7, 10..13; b = 0..15, c = 8,9,14,15; d = 0..15; e = 4..15; f = 0..15 }
<end of JIS string>:=	<1/11> <5/12> { ST (string terminator) from ISO 6429 }
<T.52 string>:=	for further study
<unicode string>:=	<uni char> <uni char> <end of unicode string> { string according to ISO 10646-1 (BMP) }
<uni char>:=	<a/b> { a = 0..15, b = 0..15 }
<end of unicode>:=	<ST version> <FF version>
<ST version>:=	<0/0> <9/12>
<FF version>:=	<F/F> <F/F>

9.3.5 NDC

This datatype denotes horizontal or vertical scalar values in the NDC space. An NDC is a fixed point real number from the range 0.0 (inclusive) to 1.0 (exclusive).

Coding rules:

- a) an NDC value is coded in a sequence of one or more bytes;
- b) bit 8 set to 1 means no extension, it is the last byte of the sequence;
- c) bit 8 set to 0 and bit 7 set to 1 indicates an extension, it is not the last byte;
- d) the encoded value is specified:
 - 1) for one byte NDC by bits 7 to 1;
 - 2) for multi-byte NDC by bits 6 to 1 from all bytes except the last byte and bits 7 to 1 from the last byte;
- e) the binary radix point is not coded, but assumed to be:
 - 1) for one byte NDC next left to the bit 7;
 - 2) for multi-byte NDC next left to bit 6 in the first byte;
- f) the most significant part of the value is coded in the first byte;
- g) the least significant part of the value is coded in the last byte.

The terminal symbol notation is: <NDC>.

Example: Coded representation of the number $0.3125_{10} = 0.0101_2$

b ₈	b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁
1	0	1	0	1	0	0	0
No extension	NDC value bits						

Example: Coded representation of the NDC $0.205078125_{10} = 0.001101001_2$

Byte 1

b ₈	b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁
0	1	0	0	1	1	0	1
Extension		Most significant bits					

Byte 2

b ₈	b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁
1	0	0	1	0	0	0	0
No extension	Least significant bits						

9.4 Attributes and lower level symbols

<autoscroll>	:=	<6/9>
<attributed text>	:=	<text> <fin reference>
<autostore>	:=	<7/8>
<bin>	:=	<INTEGER>
<bin reference>	:=	<11/8> <bin>
<bitmap disp type>	:=	<11/11> <INTEGER : 0> { <i>stretched</i> } <INTEGER : 1> { <i>centred</i> } <INTEGER : 2> { <i>tiled</i> }
<block index>	:=	<12/4> <INTEGER>
<button>	:=	<text> <bin reference>
<cin>	:=	<INTEGER>
<cin reference>	:=	<12/7> <INTEGER> { <i>reference to a component</i> }
<cin spec>	:=	<in spec>+
<closed>	:=	<6/8>
<colour>	:=	<11/10> <INTEGER> { <i>colour index</i> }
<command end>	:=	<2/15>
<defactive>	:=	<11/7> <INTEGER> { <i>the referenced cin</i> }
<default item>	:=	<12/3> <INTEGER>
<default text>	:=	<text>
<direct in>	:=	<7/10>
<display character>	:=	<a/b> { <i>a = 2..15, b = 0..15</i> }
<echo char>	:=	<14/6> <display character>
<echo type>	:=	<12/0> <INTEGER : 0> { <i>local echo</i> } <INTEGER : 1> { <i>no echo</i> } <INTEGER : 2> { <i>echo defined character</i> }
<end of list>	:=	<3/15>
<filename>	:=	<14/8> <string>
<fin>	:=	<INTEGER>
<fin reference>	:=	<10/13> <INTEGER> { <i>reference to a font resource object</i> }
<group>	:=	<12/2> <INTEGER>
<height>	:=	<9/2> <NDC>
<in spec>	:=	<list spec> <range spec> <one in> { <i>defines a list, a range or a single identification number</i> }
<list spec>	:=	<11/0> <INTEGER>+ <end of list>
<range spec>	:=	<13/0> <from> <to>
<from>	:=	<INTEGER>
<to>	:=	<INTEGER>
<one in>	:=	<13/5> <INTEGER>
<increment>	:=	<12/10> <INTEGER>
<initial font>	:=	<11/12> <INTEGER> { <i>the referenced fin</i> }
<initial value>	:=	<12/11> <INTEGER>
<input type>	:=	<11/14> <INTEGER : 0> { <i>any character</i> } <INTEGER : 1> { <i>alphabetic</i> } <INTEGER : 2> { <i>numeric</i> } <INTEGER : 3> { <i>alphanumeric</i> }
<input transform>	:=	<11/5> <INTEGER : 0> { <i>no transformation</i> } <INTEGER : 1> { <i>to lower</i> } <INTEGER : 2> { <i>to upper</i> }
<item locact>	:=	<7/13> <local action>+
<item locval>	:=	<7/14> <local action>+
<label font>	:=	<11/13> <INTEGER> { <i>the referenced fin</i> }
<label>	:=	<14/5> <string>
<list index>	:=	<12/4> <INTEGER>
<locator>	:=	<6/0>
<marked>	:=	<7/1>
<max char>	:=	<12/1> <INTEGER>

<max line>	:=	<12/5> <INTEGER>	
<max value>	:=	<12/9> <INTEGER>	
<maxim text>	:=	<7/0>	
<maximizable>	:=	<6/14>	
<min value>	:=	<12/8> <INTEGER>	
<modal>	:=	<6/12>	
<multiline>	:=	<6/15>	
<multiple choice>	:=	<7/2>	
<negative>	:=	<7/9>	
<no border>	:=	<6/13>	
<no consistency>	:=	<7/15>	
<no drop>	:=	<7/3>	
<no edit>	:=	<7/4>	
<no format>	:=	<7/6>	
<no scrolling tools>	:=	<7/5>	
<no sound>	:=	<7/11>	
<notaccessible>	:=	<6/11>	
<num blocks>	:=	<10/8> <INTEGER>	
<oin>	:=	<INTEGER>	
<oin spec>	:=	<in spec>+ { <i>defines object identifiers</i> }	
<output height>	:=	<9/4> <NDC>	
<program about>	:=	<string>	
<program description>	:=	<string>	
<program filename>	:=	<filename>	
<program name>	:=	<string>	
<program parameter>	:=	<program parameter opc> <string>	
<program type>	:=	<13/1> <INTEGER : 0> { <i>standalone program</i> }	
		<INTEGER : 1> { <i>program with filter interface</i> }	
<result>	:=	<true> <false>	
<scrolling tools>	:=	<11/1> <INTEGER : 0> { <i>terminal dependent</i> }	
		<INTEGER : 1> { <i>two column scrolling</i> }	
		<INTEGER : 2> { <i>two row scrolling</i> }	
<slider value>	:=	<INTEGER>	
<store result>	:=	<true> <false>	
<separated>	:=	<6/10>	
<sorted>	:=	<7/12>	
<template>	:=	<7/7>	
<text>	:=	<14/3> <string>	
<timestamp>	:=	<12/14> <INTEGER>	
<tin reference>	:=	<10/15> <INTEGER>	{ <i>reference to a text resource object</i> }
<title font>	:=	<11/9> <INTEGER>	{ <i>the referenced fin</i> }
<title>	:=	<14/4> <string>	
<vertical>	:=	<6/7>	
<vin reference>	:=	<10/12> <INTEGER>	
<width>	:=	<9/3> <NDC>	
<xloc>	:=	<NDC>	
<xpos>	:=	<9/0> <NDC>	
<yes/no>	:=	<true> <false>	
<ypos>	:=	<9/1> <NDC>	
<yloc>	:=	<NDC>	
<vtx byte>	:=	<a/a> { <i>a = 0..15</i> }	

9.5 Opcodes

In Tables 39 and 40, the suffix “opc” from the symbols of the command syntax specification is omitted.

The code positions that are currently not used are reserved for future extensions.

TABLE 39/T.107

Host commands opcodes

	2	3
0	open	create object
1	close	delete outdated objects
2	resume	modify component (see also 9.6)
3	suspend	obj location change
4	close all	load coltable
5	reset col	open application
6	user lock	store objects
7	user unlock	erase objects
8	open object	open blocking object
9	close object	resource transfer
10	destroy object	term cap request
11	access disable	
12	access enable	set options
13	translation mode	
14	(See Note)	
15	(See Note)	

NOTE – These code positions are reserved for the MDI.

TABLE 40/T.107

Terminal commands opcodes

	2	3
0	store objects resp	
1	object retrans	
2	open appl resp	
3	user data	
4	error	
5		
6	term cap resp	
7		
8		
9		resource transfer
10		
11		
12		
13	translation mode	
14		
15		

NOTE – Tables 41 to 43 contain symbols which have the same code. Their names are separated with a "/".

TABLE 41/T.107

Opcodes (Part 1)

	Objects	"End" codes	Components	
	2	3	4	5
0	application bar		bar menu choice	slider
1	button bar		pull-down choice	text component
2	pop-up menu		push button comp	sensitive text
3	dialogue box		pop-up choice	cas casc menu choice
4			separator	
5	operative object		frame	
6	sound object		graphic output area	
7	video object		text area	
8	message box		text input field	
9	metacode object		box push button	
10	bitmap object		check box	
11	font object		radio button	
12	text object		list box	
13	videotex object		combo box	
14		command end	sensitive area	
15		end of list		

TABLE 42/T.107

Opcodes (Part 2)

	Boolean attributes		Coordinates, dimensions
	6	7	9
0	fn bold/locator	maxim text	xpos
1	fn underline/local storage	marked	ypos
2	fn italic	multiple choice	height
3	index def	no drop	width
4	rgb def	no edit	output height
5	locactact	no scrolling tools	
6	locactval	no format	
7	vertical	template	
8	closed	autostore	
9	autoscroll	negative	
10	separated	direct in/more blocks	
11	notaccessible	no sound/store ini values	
12	modal	sorted	end of string
13	no border	item locact	
14	maximizable	item locval	
15	multiline	no consistency	

TABLE 43/T.107

Opcodes (Part 3)

	Integer attributes				String attribute
	10	11	12	13	14
0	fn family	list spec	echo type	range spec	applid/ user language
1	fn height/ data types	error oin/ scrolling tools	max char	program type	appl add data/ system information
2	fn colour/ text types	error cin	group	prev text	
3	bmcompr/ still picture	error com code	default item	current text	text
4	bits per unit	local commands	list index/ block index	next text	title
5	colour entry/ audio	input transform	max line	one in	label
6	moving picture				echo char
7	audiovisual	defactive	cin reference		vtx content
8	num blocks	bin reference	min value		filename
9	message type	title font	max value		program parameter
10	lifetime	colour	increment		current block
11	default vtx	bitmap disptype	initial value		
12	vin reference	initial font	max time		basic page #
13	fin reference	label font	input timeout		
14		input type	timestamp		
15	tin reference				

9.6 Syntax of the VEMMI_Modify_Component

VEMMI_Modify_Component is used to modify or to add one or more components in an object. This command has the same syntax as VEMMI_Create_Object, except that it contains only those attributes which are to be modified. All attributes which are modifiable or their higher symbol definitions are summarized in the following:

autoscroll	echo type	list text unit	no sound
attributed text	group	locactval	notaccessible
bin reference	increment	locactact	previous text
cin reference	initial font	max char	text
closed	initial info	max value	tin reference
colour	input type	marked	title
current text	intext	message	
default item	label	min value	
default text	label font	negative	
echo char	lifetime	next text	

The modification of boolean attributes is achieved by transmitting the same attribute again. The host application can either set or unset the value for a boolean attribute.

In order to set a boolean attribute, the host application shall transmit the attribute once (this is equivalent to the object creation syntax). In order to unset a boolean attribute, the host application shall transmit the attribute twice immediately following each other.

In the following example a pop-up menu offers four choices to the user, two accessible (CINs 35, 36) and two not accessible (CINs 37, 38):

<u>S</u> tart New Game
<u>B</u> eginner Level
<u>I</u> ntermediate Level
<u>A</u> dvanced Level

Upon reception of VEMMI_Modify_Component with:

```
<oin : 2>  
<cin : 37> <NotAccessible> <NotAccessible>  
<cin : 39> <text : "&Double Cube"> <separated>
```

the menu is changed, one inaccessible component is made accessible and a new component is added.

<u>S</u> tart New Game
<u>B</u> eginner Level
<u>I</u> ntermediate Level
<u>A</u> dvanced Level
<u>D</u> ouble Cube

For the optional attributes the default values are applied, if VEMMI_Modify_Component adds a new component. That is the same rule as VEMMI_Create_Object uses. If VEMMI_Modify_Component references an already existing component, only the transmitted attributes are applied. The missing attributes remain unchanged.

VEMMI_Modify_Component shall not be applied to the components of the application bar.

Specific requirements for the modification of list items are described in 7.6.1.11 and 7.6.1.12.

9.7 Defaults

The default values of attributes are applied when an optional attribute is not contained in the element definition. An exception makes VEMMI_Modify_Component applied on existing components, see 9.6. The default values depend on the attribute datatype and on the object/component to which they belong.

For the default value of a Boolean attribute (these are coded with the opcode only) the logical negative meaning compared with its definition shall be assumed. A default value of a Boolean attribute is not codeable and is never contained in the element definition. See Tables 44 to 48.

TABLE 44/T.107

Boolean attributes defaults

Attribute	Default value	Attribute	Default value
fn bold	normal, non-bold	maxim text	no maximised text
fn underline	not underlined	marked	unmarked
fn italic	non-italic	multiple choice	single choice
vertical	horizontal	no drop	drop
closed	open	no edit	editable
notaccessible	accessible	no scrolling tools	with scrolling tools
modal	non-modal	no format	formatable
no border	with border		
autoscroll	no autoscroll	template	no "template" store
separated	not separated	autostore	no "autostore"
multiline	single line	no sound	with sound
no consistency	with consistency	sorted	not sorted
maximizable	not maximizable	negative	positive
		direct in	no direct input

TABLE 45/T.107

Integer attributes defaults

Attribute	Default value	Attribute	Default value
fn family	SWISS	input type	0
fn height	10	echo type	0
fn colour	0 (black)	max char	1
bits per units (pixel)	1	group	1
bits per units (component)	8	default item	1
		list index	1 for the first, all others are numbered consecutively
message type	0	max line	terminal dependent
lifetime	1	sound format	0
colour entry	0	initial value	min value
		increment	1
defactive	1	max time	10
title font	default font	input timeout	infinite
colour	0 (black)		
bitmap disptype	stretched		
initial font	default font		
label font	default font		

TABLE 46/T.107

NDC attributes defaults

Attribute	Default value
xpos, ypos of the message box	terminal dependent
xpos	0.0
ypos	0.0

TABLE 47/T.107

Width and height defaults

Element	Width default value	Height default value
application bar	terminal dependent	terminal dependent
button: button bar	0.125	0.03
pop-up menu	terminal dependent	
dialogue box	1.0	0.75
separator	dialogue box width	dialogue box height
frame	dialogue box width	dialogue box height
text area	dialogue box width	dialogue box height
graphic output area	dialogue box width	dialogue box height
text input field	dialogue box width	0.03
push button	0.125	0.03
check box	0.125	0.03
radio button	0.125	0.03
list box	dialogue box width	dialogue box height
combination box	dialogue box width	dialogue box height
sensitive area	0.125	0.03
message box	terminal dependent	terminal dependent

TABLE 48/T.107

String attributes defaults

Attribute	Default value
echo char	2/13

10 Introduction of the VEMMI service into existing Videotex Recommendations

10.1 Introduction of the VEMMI to T.101 [4]

The VEMMI Protocol Elements are mapped on the Videotex Presentation Data Elements (VPDEs).

In order to enable the data flow between Videotex terminal and VEMMI application, the logical channel shall be transparent for VEMMI data blocks.

10.2 Introduction of the VEMMI to T.105 [6]

The VEMMI Protocol Elements are mapped on one or more SBV_VTX_Data service element.

In order to enable the data flow between Videotex terminal and VEMMI application, the logical channel shall be transparent for VEMMI data blocks.

Annex A

T.51String

(This annex forms an integral part of this Recommendation)

A.1 Scope

This annex describes the rules to be applied when encoding names, strings, etc. using Recommendation T.51 [2]. The T.51String defined by this annex serves as a reference model by restricting Recommendation T.51 [2] to those elements of the code extension mechanisms, of the character sets and repertoire which are necessary to ease the implementations.

This definition is not specific to an individual telematic service, but it can be referenced by telematic application standards.

A.2 Graphic character sets

The primary set of graphic characters (see Figure A.1) is identical with the set of graphic characters of the International Reference Version (IRV) of the 7-bit coded character set of Recommendation T.50 [1].

				b ₇	0	0	0	0	1	1	1	1	
				b ₆	0	0	1	1	0	0	1	1	
				b ₅	0	1	0	1	0	1	0	1	
					0	1	2	3	4	5	6	7	
b ₄	b ₃	b ₂	b ₁										
0	0	0	0	0				0	@	P	.	p	
0	0	0	1	1				!	1	A	Q	a	q
0	0	1	0	2				"	2	B	R	b	r
0	0	1	1	3				#	3	C	S	c	s
0	1	0	0	4				\$	4	D	T	d	t
0	1	0	1	5				%	5	E	U	e	u
0	1	1	0	6				&	6	F	V	f	v
0	1	1	1	7				'	7	G	W	g	w
1	0	0	0	8				(8	H	X	h	x
1	0	0	1	9)	9	I	Y	i	y
1	0	1	0	10				*	:	J	Z	j	z
1	0	1	1	11				+	;	K	[k	{
1	1	0	0	12				,	<	L	\	l	
1	1	0	1	13				-	=	M]	m	}
1	1	1	0	14				.	>	N	^	n	~
1	1	1	1	15				/	?	O	_	o	

T0821490-95/d42

FIGURE A.1/T.107
Primary set of graphic characters for T.51 String

The supplementary set of graphic characters is specified in Figure A.2.

					b7	0	0	0	0	1	1	1	1
					b6	0	0	1	1	0	0	1	1
					b5	0	1	0	1	0	1	0	1
						0	1	2	3	4	5	6	7
b4	b3	b2	b1		0	1	2	3	4	5	6	7	
0	0	0	0	0			NBSP	o		—	Ω	K	
0	0	0	1	1			i	±	·	1	Æ	æ	
0	0	1	0	2			ç	²	·	®	Ð	ð	
0	0	1	1	3			£	³	^	©	a	ø	
0	1	0	0	4				x	~	™	†	‡	
0	1	0	1	5			¥	μ	—	♯			
0	1	1	0	6				¶	˘	¬	IJ	ij	
0	1	1	1	7			§	·	·	¡	Ł	ł	
1	0	0	0	8			¤	÷	..		Ł	ł	
1	0	0	1	9			·	·			Ø	ø	
1	0	1	0	10			“	”	°		Œ	œ	
1	0	1	1	11			«	»	¸		o	ß	
1	1	0	0	12			←	¼		⅛	Ɔ	ɔ	
1	1	0	1	13			↑	½	”	⅜	Ɔ	ɔ	
1	1	1	0	14			→	¾	˘	⅝	Ŋ	ŋ	
1	1	1	1	15			↓	¿	˘	⅞	'n	SHY	

T0821500-95/d43


 These code positions shall not be used

FIGURE A.2/T.107
 Supplementary set of graphic characters for T.51 String

For the T.51String the following applies:

- 1) The primary set is designated as the G0 set and invoked in column 2 to 7 of the code table.
- 2) The supplementary set is designated as a G2 set. In the 8-bit environment the set is invoked in column 10 to 15.
- 3) In the T.51String no designation and invocation sequences are allowed.
- 4) All characters in column 4 of the supplementary set are non-spacing characters (diacritical marks).
- 5) Unallocated code positions are reserved and shall not be used.
- 6) The compatibility provisions described in 2.2.4/T.51 [2] Notes 3, 4 and 6 are not applied.

A.3 Code extension technique

In the 8-bit environment no code extension sequence is allowed. In the 7-bit environment the single shift function SS2 (code 1/9) is used to invoke one character from the G2 set. All other shift functions are not allowed.

A.4 Repertoire of the Latin based character set

The T.51String repertoire is identical with the superset of the repertoire of the Latin based character set specified in Annex A/T.51 [2]. All combinations of diacritical marks with basic letters as specified in Figure A.2/T.51 [2] are supported.

The Notes and remarks from A.4/T.51 [2] shall not apply.

A.5 Control functions

Invocation and designation sequences for control functions are not allowed. From columns 0 and 1 of the code table in use only the control characters CR, Carriage Return (0/13) and LF, Line Feed (0/10) can be used. They are specified in Recommendation T.50 [1].

Annex B

Mandatory subset of ISO 8859 [13]

(This annex does not form an integral part of this Recommendation)

	2	3	4	5	6	7		10	11	12	13	14	15
0	SP	0	@	P	'	p							
1	!	1	A	Q	a	q							
2	"	2	B	R	b	r							
3	#	3	C	S	c	s							
4	\$	4	D	T	d	t							
5	%	5	E	U	e	u							
6	&	6	F	V	f	v							
7	'	7	G	W	g	w							
8	(8	H	X	h	x							
9)	9	I	Y	i	y							
10	*	:	J	Z	j	z							
11	+	;	K	[k	{							
12	,	<	L	\	l								
13	-	=	M]	m	}							
14	.	>	N	^	n	~							
15	/	?	O	_	o								

FIGURE B.1/T.107

Mandatory subset of ISO 8859 [13] code table

Annex C

Minimum datatype kernel

(This annex forms an integral part of this Recommendation)

A VEMMI terminal shall support the following minimum datatype kernel.

TABLE C.1/T.107

Supported datatypes

Class	Encoding standard	Remarks
For text data		
	VEMMI high quality text	
	ISO 8859 [13] (fully-formed accented characters) from code position 2/0 to code position 7/15 as given in Annex B	For Latin based terminal
	Shift JIS Code (for Japanese characters)	For Japanese terminals
For still picture		
	ISO 10918 [16] (JPEG)	
	VEMMI device independent bitmap	
	Graphics Interchange Format (sm) Version 89a. CompuServe Incorporated Columbus, Ohio, USA (only the functions of Version 87 must be supported)	
	Microsoft Windows Device-Independent Bitmap (DIB) with RLE4, RLE8 and 1, 4, 8 or 24 bits per pixel	
For audio data		
	WAVE format	(See Note)
	MIDI format	(See Note)
NOTE – Mandatory in this case denotes that a terminal shall support all VEMMI services related to the functionality. It does not mean that a terminal shall be equipped with the necessary device to play this data.		