

INTERNATIONAL STANDARD

NORME INTERNATIONALE



**Function blocks (FB) for process control and electronic device description language (EDDL) –
Part 4: EDD interpretation**

**Blocs fonctionnels (FB) pour les procédés industriels et le langage de description électronique de produit (EDDL) –
Partie 4: Interprétation EDD**



THIS PUBLICATION IS COPYRIGHT PROTECTED

Copyright © 2015 IEC, Geneva, Switzerland

All rights reserved. Unless otherwise specified, 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 either IEC or IEC's member National Committee in the country of the requester. If you have any questions about IEC copyright or have an enquiry about obtaining additional rights to this publication, please contact the address below or your local IEC member National Committee for further information.

Droits de reproduction réservés. Sauf indication contraire, aucune partie de cette publication ne peut être reproduite ni utilisée sous quelque forme que ce soit et par aucun procédé, électronique ou mécanique, y compris la photocopie et les microfilms, sans l'accord écrit de l'IEC ou du Comité national de l'IEC du pays du demandeur. Si vous avez des questions sur le copyright de l'IEC ou si vous désirez obtenir des droits supplémentaires sur cette publication, utilisez les coordonnées ci-après ou contactez le Comité national de l'IEC de votre pays de résidence.

IEC Central Office
3, rue de Varembe
CH-1211 Geneva 20
Switzerland

Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00
info@iec.ch
www.iec.ch

About the IEC

The International Electrotechnical Commission (IEC) is the leading global organization that prepares and publishes International Standards for all electrical, electronic and related technologies.

About IEC publications

The technical content of IEC publications is kept under constant review by the IEC. Please make sure that you have the latest edition, a corrigenda or an amendment might have been published.

IEC Catalogue - webstore.iec.ch/catalogue

The stand-alone application for consulting the entire bibliographical information on IEC International Standards, Technical Specifications, Technical Reports and other documents. Available for PC, Mac OS, Android Tablets and iPad.

IEC publications search - www.iec.ch/searchpub

The advanced search enables to find IEC publications by a variety of criteria (reference number, text, technical committee,...). It also gives information on projects, replaced and withdrawn publications.

IEC Just Published - webstore.iec.ch/justpublished

Stay up to date on all new IEC publications. Just Published details all new publications released. Available online and also once a month by email.

Electropedia - www.electropedia.org

The world's leading online dictionary of electronic and electrical terms containing more than 30 000 terms and definitions in English and French, with equivalent terms in 15 additional languages. Also known as the International Electrotechnical Vocabulary (IEV) online.

IEC Glossary - std.iec.ch/glossary

More than 60 000 electrotechnical terminology entries in English and French extracted from the Terms and Definitions clause of IEC publications issued since 2002. Some entries have been collected from earlier publications of IEC TC 37, 77, 86 and CISPR.

IEC Customer Service Centre - webstore.iec.ch/csc

If you wish to give us your feedback on this publication or need further assistance, please contact the Customer Service Centre: csc@iec.ch.

A propos de l'IEC

La Commission Electrotechnique Internationale (IEC) est la première organisation mondiale qui élabore et publie des Normes internationales pour tout ce qui a trait à l'électricité, à l'électronique et aux technologies apparentées.

A propos des publications IEC

Le contenu technique des publications IEC est constamment revu. Veuillez vous assurer que vous possédez l'édition la plus récente, un corrigendum ou amendement peut avoir été publié.

Catalogue IEC - webstore.iec.ch/catalogue

Application autonome pour consulter tous les renseignements bibliographiques sur les Normes internationales, Spécifications techniques, Rapports techniques et autres documents de l'IEC. Disponible pour PC, Mac OS, tablettes Android et iPad.

Recherche de publications IEC - www.iec.ch/searchpub

La recherche avancée permet de trouver des publications IEC en utilisant différents critères (numéro de référence, texte, comité d'études,...). Elle donne aussi des informations sur les projets et les publications remplacées ou retirées.

IEC Just Published - webstore.iec.ch/justpublished

Restez informé sur les nouvelles publications IEC. Just Published détaille les nouvelles publications parues. Disponible en ligne et aussi une fois par mois par email.

Electropedia - www.electropedia.org

Le premier dictionnaire en ligne de termes électroniques et électriques. Il contient plus de 30 000 termes et définitions en anglais et en français, ainsi que les termes équivalents dans 15 langues additionnelles. Egalement appelé Vocabulaire Electrotechnique International (IEV) en ligne.

Glossaire IEC - std.iec.ch/glossary

Plus de 60 000 entrées terminologiques électrotechniques, en anglais et en français, extraites des articles Termes et Définitions des publications IEC parues depuis 2002. Plus certaines entrées antérieures extraites des publications des CE 37, 77, 86 et CISPR de l'IEC.

Service Clients - webstore.iec.ch/csc

Si vous désirez nous donner des commentaires sur cette publication ou si vous avez des questions contactez-nous: csc@iec.ch.

INTERNATIONAL STANDARD

NORME INTERNATIONALE



**Function blocks (FB) for process control and electronic device description
language (EDDL) –
Part 4: EDD interpretation**

**Blocs fonctionnels (FB) pour les procédés industriels et le langage de
description électronique de produit (EDDL) –
Partie 4: Interprétation EDD**

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

COMMISSION
ELECTROTECHNIQUE
INTERNATIONALE

ICS 25.040.40; 35.240.50

ISBN 978-2-8322-2937-8

**Warning! Make sure that you obtained this publication from an authorized distributor.
Attention! Veuillez vous assurer que vous avez obtenu cette publication via un distributeur agréé.**

CONTENTS

FOREWORD.....	8
INTRODUCTION.....	10
1 Scope.....	11
2 Normative references	11
3 Terms, definitions, abbreviated terms, acronyms and conventions.....	11
3.1 General terms and definitions	12
3.2 Terms and definitions related to modular devices.....	12
3.3 Abbreviated terms and acronyms	13
3.4 Conventions.....	13
4 EDDL user interface description	13
4.1 Overview.....	13
4.2 Menu conventions for handheld applications	14
4.3 Menu conventions for PC-based applications	14
4.3.1 Overview	14
4.3.2 Online Root Menus	14
4.3.3 Offline Root Menu.....	15
4.3.4 Example of EDD menu structure.....	15
4.3.5 User interface.....	20
4.4 Containers and contained items.....	23
4.4.1 Overview	23
4.4.2 Containers.....	23
4.4.3 Contained items.....	26
4.5 Layout rules.....	30
4.5.1 Overview	30
4.5.2 Layout rules for WIDTH and HEIGHT.....	30
4.5.3 Layout rules for COLUMNBREAK and ROWBREAK.....	31
4.5.4 Layout examples	37
4.5.5 Conditional user interface.....	45
4.6 Graphical elements.....	46
4.6.1 Overview	46
4.6.2 Graph and chart.....	47
4.6.3 Common attributes	47
4.6.4 CHART	48
4.6.5 GRAPH.....	56
4.6.6 AXIS.....	65
4.6.7 IMAGE.....	66
4.6.8 GRID	67
5 EDDL data description.....	69
5.1 Variables	69
5.1.1 VARIABLE TYPEs	69
5.1.2 VARIABLE CLASS.....	70
5.1.3 VARIABLE ACTIONS.....	70
5.2 EDDL application stored device data.....	70
5.2.1 Overview	70
5.2.2 FILE	71
5.2.3 LIST	73

5.3	Exposing data items outside the EDD application.....	80
5.4	Initialization of EDD instances.....	80
5.4.1	Overview	80
5.4.2	Initialization support	80
5.4.3	TEMPLATE.....	80
5.5	Device model mapping.....	81
5.5.1	BLOCK_A.....	81
5.5.2	BLOCK_B.....	82
6	EDDL METHOD programming and usage of Builtins	82
6.1	Builtin MenuDisplay	82
6.2	Division by zero and undetermined floating values	85
6.2.1	Integer and unsigned integer values	85
6.2.2	Floating-point values	85
7	Modular devices	85
7.1	Overview.....	85
7.2	EDD identification	86
7.3	Instance object model	86
7.4	Offline configuration.....	87
7.5	Online configuration.....	87
7.6	Simple modular device example.....	87
7.6.1	General	87
7.6.2	Separate EDD file example with direct EDD referencing	88
7.6.3	Separate EDD file example with classification EDD referencing and interfaces	89
7.6.4	One EDD file example	92
7.6.5	Combination of single and separate modular device example	93
7.7	COMPONENT_RELATION.....	93
7.7.1	General	93
7.7.2	NEXT_COMPONENT usage	93
7.7.3	REQUIRED_RANGES and ADDRESSING usage.....	93
7.8	Upload and download for modular devices	93
7.9	Diagnostic.....	94
7.10	Reading modular device topology	95
7.10.1	SCAN	95
7.10.2	Detect module type.....	96
7.11	Configuration check	97
8	Edit session.....	98
8.1	Data management.....	98
8.1.1	Overview	98
8.1.2	General rules.....	99
8.1.3	Data caching for dialogs and windows	99
8.1.4	Data caching for METHODS.....	100
8.2	UI aspects of editing sessions.....	102
8.3	User roles	103
9	Offline and online configuration	103
9.1	Overview.....	103
9.2	Offline dataset	104
9.3	Offline configuration.....	104
9.4	Online dataset	104

9.5	Online configuration	104
9.6	Upload and download	105
9.6.1	Overview	105
9.6.2	Error recovery.....	106
9.6.3	Upload procedure	106
9.6.4	Download procedure	107
10	EDDL communication description	109
10.1	COMMAND	109
10.1.1	General	109
10.1.2	OPERATION.....	109
10.1.3	TRANSACTION	110
10.1.4	Command addressing	113
10.2	Parsing data received from the device	114
10.2.1	General	114
10.2.2	Parsing complex data items.....	114
10.2.3	FOUNDATION Fieldbus	114
10.2.4	HART	115
10.2.5	PROFIBUS and PROFINET	115
10.3	FOUNDATION Fieldbus communication model.....	115
11	EDD development.....	119
11.1	Dictionaries.....	119
11.2	Reserved	119
Annex A	(normative) Device simulation.....	120
Annex B	(informative) Predefined identifiers	121
Figure 1	– EDD example of root menus.....	20
Figure 2	– Example of an EDD application for diagnostics	20
Figure 3	– Example of an EDD application for process variables.....	21
Figure 4	– Example of an EDD application for primary variables	21
Figure 5	– Example of an EDD application for process-related device features	22
Figure 6	– Example of an EDD application for device features	22
Figure 7	– Example of an EDD application for maintenance features	23
Figure 8	– Usage of COLLECTION MEMBERS in MENUs of STYLE GROUP	26
Figure 9	– Displaying single bits of BIT_ENUMERATED	27
Figure 10	– Displaying multiple bits of BIT_ENUMERATED.....	28
Figure 11	– Example of an EDD application for a variable of type BIT_ENUMERATED	28
Figure 12	– EDD source code for layout for protruding elements example.....	32
Figure 13	– Layout for protruding elements.....	32
Figure 14	– EDD source code for layout for partially filled rows example.....	33
Figure 15	– Layout for partially filled rows.....	33
Figure 16	– EDD source code for layout for partially filled rows example.....	34
Figure 17	– Layout for partially filled rows.....	34
Figure 18	– EDD source code for layout for oversized elements example.....	35
Figure 19	– Layout for oversized elements.....	35
Figure 20	– EDD source code example for a layout for columns in stacked group	36
Figure 21	– Layout for columns in stacked group	36

Figure 22 – EDD source code for layout for columns with GRAPHS in stacked group example	37
Figure 23 – Layout for columns with GRAPHS in stacked group	37
Figure 24 – Example of an EDD for an overview menu	37
Figure 25 – Example of an EDD application for an overview window	38
Figure 26 – Example of an EDD using COLUMNBREAK	38
Figure 27 – Example of an EDD application for an overview window	39
Figure 28 – EDD example for an overview window	39
Figure 29 – Example of an EDD application for an overview window	40
Figure 30 – Example of an EDD for in-line graphs and charts	40
Figure 31 – Example of an EDD application for an in-line graph	41
Figure 32 – Example of an EDD for full-width graphs and charts	41
Figure 33 – Example of an EDD application for a full-width graph	42
Figure 34 – Example of an EDD for nested containers	43
Figure 35 – Example of an EDD application for nested containers	43
Figure 36 – Example of an EDD for EDIT_DISPLAYS	44
Figure 37 – Example of an EDD application for EDIT_DISPLAYS	44
Figure 38 – Example of an EDD for images	45
Figure 39 – Example of an EDD application for images	45
Figure 40 – HEIGHT and WIDTH attributes for CHART and GRAPH	47
Figure 41 – EMPHASIS attribute to differentiate one or more SOURCES or WAVEFORMs	48
Figure 42 – Example of a chart with one curve in a dialog	50
Figure 43 – Example of a chart with two SOURCES	51
Figure 44 – Displaying example of a chart with two SOURCES	52
Figure 45 – Example of a chart with three horizontal bars	53
Figure 46 – Displaying example of a chart with three horizontal bars	54
Figure 47 – Example of a chart in a dialog	56
Figure 48 – A graph and the visual elements	57
Figure 49 – Example of a graph	60
Figure 50 – Multiple used axes	61
Figure 51 – EDD with device-supported zooming and scrolling	65
Figure 52 – EDD example of an IMAGE	66
Figure 53 – EDD example of an IMAGE with the LINK attribute	66
Figure 54 – EDD example of a GRID	68
Figure 55 – Result of the EDD example	68
Figure 56 – Wrong usage of a BIT_ENUMERATED variable	69
Figure 57 – Usage of ENUMERATED instead of BIT_ENUMERATED	69
Figure 58 – Example of a file declaration	72
Figure 59 – Example of comparing valve signatures	73
Figure 60 – Example of more complex file declaration	74
Figure 61 – Example of reviewing the stored radar signals	75
Figure 62 – Example of an EDD that inserts, replaces, or compares radar signals	80
Figure 63 – Example of TEMPLATE usage	81

Figure 64 – Example of a BLOCK_A	82
Figure 65 – Example of a wizard	84
Figure 66 – The different relations of a module	87
Figure 67 – Components and possible configuration of the modular devices	87
Figure 68 – Separate EDD file example with direct EDD referencing	88
Figure 69 – EDD example for module1	89
Figure 70 – EDD example for module2	89
Figure 71 – EDD example for modular device	90
Figure 72 – EDD example for module1	91
Figure 73 – EDD example for module2	91
Figure 74 – EDD example for module2	93
Figure 75 – NEXT_COMPONENT usage	93
Figure 76 – REQUIRED_RANGES usage	93
Figure 77 – Upload/download order of a modular device	94
Figure 78 – Example of a SCAN METHOD	96
Figure 79 – Example of a DETECT METHOD	97
Figure 80 – Example of a CHECK_CONFIGURATION METHOD	98
Figure 81 – Data caching for an offline session	98
Figure 82 – Data caching for an online session	99
Figure 83 – Sub dialogs or windows using a shared edit cache	100
Figure 84 – Sub dialogs or windows using separate edit caches	100
Figure 85 – Data caching for nested METHODS	101
Figure 86 – Data caching for a METHOD invoked within a dialog	101
Figure 87 – Data caching for a METHOD invoking a dialog using an edit cache	101
Figure 88 – Data caching for a METHOD invoking a dialog	102
Figure 89 – Data flow for download to the device	105
Figure 90 – Data flow for upload from the device	105
Figure 91 – Example of a single item mask	111
Figure 92 – Mapping example with a single item mask	111
Figure 93 – Multiple item masks	111
Figure 94 – Mapping example with a multiple item mask	112
Figure 95 – INFO qualifier	112
Figure 96 – INDEX qualifier	113
Figure 97 – INFO and INDEX qualifier	113
Figure 98 – Example device with 2 unique BLOCK_A definitions	116
Figure 99 – Example EDD for a device with 2 unique BLOCK_A definitions	117
Figure 100 – BLOCK_A example with PARAMETER_LISTS	118
Figure 101 – Example EDD for a BLOCK_A with PARAMETER_LISTS	119
Table 1 – List of defined root menu identifiers for handhelds	14
Table 2 – List of defined root menu identifiers for PC-based devices	14
Table 3 – Fall back alternatives for online root menus	15
Table 4 – Fall back alternatives for offline root menus	15
Table 5 – Permitted contained items and default STYLES	24

Table 6 – WIDTH and HEIGHT span and applicability	31
Table 7 – Image formats	67
Table 8 – String handling	69
Table 9 – Examples of floating-point results	85
Table 10 – Usages of COMPONENT_PATH	86
Table 11 – Diagnostic classifications	95
Table 12 – Builtins for method cache controlling	102
Table 13 – List of defined upload menu identifiers	106
Table 14 – List of defined download menu identifiers	107
Table 15 – PROFIBUS and PROFINET communication mapping.....	110
Table B.1 – Predefined identifiers	121

INTERNATIONAL ELECTROTECHNICAL COMMISSION

**FUNCTION BLOCKS (FB) FOR PROCESS CONTROL AND
ELECTRONIC DEVICE DESCRIPTION LANGUAGE (EDDL) –****Part 4: EDD interpretation****FOREWORD**

- 1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
- 2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
- 3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
- 4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
- 5) IEC itself does not provide any attestation of conformity. Independent certification bodies provide conformity assessment services and, in some areas, access to IEC marks of conformity. IEC is not responsible for any services carried out by independent certification bodies.
- 6) All users should ensure that they have the latest edition of this publication.
- 7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.
- 8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.
- 9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

International Standard IEC 61804-4 has been prepared by subcommittee 65E: Devices and integration in enterprise systems, of IEC technical committee 65: Industrial-process measurement, control and automation.

This first edition cancels and replaces IEC TR 61804-4 published in 2006. This edition constitutes a technical revision.

This edition includes the following significant technical changes with respect to the previous edition:

- New paragraph:
 - EDDL data description
 - EDDL METHOD programming and usage of builtins
 - Edit session
 - Offline and online configuration

- EDDL communication description
- Enhancements in paragraph EDDL user interface descriptions

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

FDIS	Report on voting
65E/465/FDIS	65E/481/RVD

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

This publication has been drafted in accordance with the ISO/IEC Directives, Part 2.

A list of all parts in the IEC 61804 series, published under the general title *Function blocks (FB) for process control and electronic device description language (EDDL)*, can be found on the IEC website.

Future standards in this series will carry the new general title as cited above. Titles of existing standards in this series will be updated at the time of the next edition.

The committee has decided that the contents of this publication will remain unchanged until the stability date indicated on IEC web site under "<http://webstore.iec.ch>" in the data related to the specific publication. At this date, the publication will be

- reconfirmed,
- withdrawn,
- replaced by a revised edition, or
- amended.

IMPORTANT – The 'colour inside' logo on the cover page of this publication indicates that it contains colours which are considered to be useful for the correct understanding of its contents. Users should therefore print this document using a colour printer.

INTRODUCTION

This part of IEC 61804 was developed using material from FDI Cooperation LLC (Foundation™ Fieldbus¹, HART®² Communication Foundation (HCF), PROFIBUS™³ Nutzerorganisation e.V. (PNO)), OPC Foundation (OPCF) and FDT Group. IEC 61804 has the general title "Function blocks (FB) for process control and Electronic Device Description Language (EDDL)".

This edition does reflect many of the various rules defined by the different communication foundations, however it is not a complete representation of those rules defined by each of the communication foundations today. Therefore, an EDD application and EDD developer will need to rely on both IEC 61804-4 and the respective communication foundation documents (e.g. specifications, test requirements, test cases) to develop a conformant application that will meet foundation registration requirements.

Conformity assessment of an EDD application is the responsibility of the respective communication foundations. In cases of any ambiguity, the rules of the respective communication foundations apply.

This part of IEC 61804

- contains an overview of the use of EDDL;
- provides examples demonstrating the use of the EDDL constructs;
- shows how the use cases are fulfilled; and
- shows the proper EDD application interpretation for each example.

This part of IEC 61804 is not an EDDL tutorial and is not intended to replace the EDDL specification.

Instructions are provided for the EDD application, which describe what will be performed without prescribing the technology used in the host implementation. For example, the FILE construct describes data that is stored by the EDD application on behalf of the EDD. The FILE construct does not specify how the data is stored. The EDD application can use a database, a flat file, or any other implementation it chooses.

¹ FOUNDATION™ Fieldbus is the trademark of the Fieldbus Foundation. This information is given for the convenience of users of this document and does not constitute an endorsement by IEC of the product named. Equivalent products may be used if they can be shown to lead to the same results.

² HART® is the registered trademark of the HART Communication Foundation. This information is given for the convenience of users of this document and does not constitute an endorsement by IEC of the product named. Equivalent products may be used if they can be shown to lead to the same results.

³ PROFIBUS and PROFINET are the trademarks of the PROFIBUS Nutzerorganisation e.V. This information is given for the convenience of users of this document and does not constitute an endorsement by IEC of the product named. Equivalent products may be used if they can be shown to lead to the same results.

FUNCTION BLOCKS (FB) FOR PROCESS CONTROL AND ELECTRONIC DEVICE DESCRIPTION LANGUAGE (EDDL) –

Part 4: EDD interpretation

1 Scope

This part of IEC 61804 specifies EDD interpretation for EDD applications and EDDs to support EDD interoperability. This document is intended to ensure that field device developers use the EDDL constructs consistently and that the EDD applications have the same interpretations of the EDD. It supplements the EDDL specification to promote EDDL application interoperability and improve EDD portability between EDDL applications.

2 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IEC 61784-1, *Industrial communication networks – Profiles – Part 1: Fieldbus profiles*

IEC 61784-2, *Industrial communication networks – Profiles – Part 2: Additional fieldbus profiles for real-time networks based on ISO/IEC 8802-3*

IEC 61804-2, *Function blocks (FB) for process control – Part 2: Specification of FB concept*

IEC 61804-3⁴, *Function blocks (FB) for process control and Electronic device description language (EDDL) – Part 3: EDDL syntax and semantics*

IEC 61804-5⁵, *Function blocks (FB) for process control and Electronic device description language (EDDL) – Part 5: EDDL Builtin library*

ISO/IEC 10918 (all parts), *Information technology – Digital compression and coding of continuous-tone still images*

ISO/IEC 15948, *Information technology – Computer graphics and image processing – Portable Network Graphics (PNG): Functional specification*

3 Terms, definitions, abbreviated terms, acronyms and conventions

For the purposes of this document, the terms and definitions given in IEC 61804-3 as well as the following apply.

⁴ To be published.

⁵ To be published.

3.1 General terms and definitions

3.1.1

EDD developer

individual or team that develops an EDD

3.1.2

container

user interface elements that contain other user interface elements

Note 1 to entry: Containers can include menus, windows, dialogs, tables, pages, groups, and other containers.

3.1.3

contained item

user interface elements that can be contained in containers

Note 1 to entry: Contained items can include variables, methods, graphs, charts, images, static text.

3.1.4

device developer

individual or team that develops a device and an EDD that describes the device

3.1.5

handheld

device with limited display resolution that restricts EDD applications user interface

3.2 Terms and definitions related to modular devices

3.2.1

channel

connection to a process that is being measured or controlled

3.2.2

component

software or hardware item contained within the modular device concept

Note 1 to entry: A component cannot function separately from a modular device hosting it. A component may support one or more types of modular devices.

3.2.3

interface

basic declarations of basic constructs

Note 1 to entry: An interface defines all public parts that components may use.

3.2.4

modal window

a child window that requires users to interact with it before they can return to operating the parent application, thus preventing the workflow on the application main window

3.2.5

modular device

device that can contain a variety of software and or hardware components

3.3 Abbreviated terms and acronyms

CP	Communication Profile
CPF	Communication Profile Family
EDD	Electronic Device Description
EDDL	Electronic Device Description Language
PC	Personal computer
PI	PROFIBUS and PROFINET International
PI PROFILE PA	PI specific profile for Process Automation field devices

3.4 Conventions

There exists some differences using and interpreting EDDL based on the used communication network and device model. The different communication networks used within this part of IEC 61804 are:

- HART (according to IEC 61784-1 CPF 9)
- FOUNDATION fieldbus (according to IEC 61784-1 CPF 1)
- PROFIBUS (according to IEC 61784-1 CPF 3)
- PROFINET (according to IEC 61784-2 CPF 3)

EDD examples in this standard show parts of EDD implementations and are incomplete.

EDDL keywords are written in upper case letters. If more than one basic construct item is meant the keyword is written in uppercase letters added with a lower case 's' for example VARIABLES.

The capitalized word Builtin refers to functions specified in IEC 61804-5.

EXAMPLE Builtin MenuDisplay refers to the Builtin named MenuDisplay specified in IEC 61804-5.

4 EDDL user interface description

4.1 Overview

Most EDD applications can be characterized as either a PC application or a handheld application. Due to the relatively small screen of a handheld device, handheld applications can only display a small amount of information at any given time. On the other hand, PC applications can provide a much more beneficial user interface, largely due to their larger screen size.

To support the capabilities of PC applications, the MENU construct has been extended in IEC 61804-3 compared to previous definitions in IEC 61804-2. Due to the differences in the user interfaces of PC applications and handheld applications, it is expected that many devices will define two MENU hierarchies – one for handheld applications and the other for PC applications. Some MENUs may be used in both hierarchies. Therefore, the entire hierarchy does not need to be specified twice.

Different menu structures for different classes of applications are possible. This standard shall be used to create menu structures in an EDD that are interpreted by applications in an unambiguous way. To provide interoperability across applications, this standard shall be followed.

4.2 Menu conventions for handheld applications

EDD applications use specific menus in the EDDs to show the user interface of the device (see Table 1). In addition user interface items can be described for handhelds and PCs at once. For handhelds strings can have beside a language code, a specific country code zz to specify shorter strings or lower resolution images (see IEC 61804-3).

Table 1 – List of defined root menu identifiers for handhelds

Menu identifier	Default STYLE	Short description
root_menu	TABLE	Handheld root menu for HART devices. This is mandatory for all HART EDDs
Menu_Top*	MENU	Handheld root menu prefix for BLOCK_A MENU_ITEMS for FOUNDATION fieldbus devices e.g. Menu_Top_TB

4.3 Menu conventions for PC-based applications

4.3.1 Overview

EDD applications use special menus in the EDDs to show the user interface of the device. Such menus are defined for diagnostic, process variables, device features, and offline configuration. Table 2 defines identifiers for the different root menus with its default STYLES. The default STYLE is used by the EDD application if the STYLE attribute is not defined in the MENU. The “Usage” column in Table 2 defines how the EDD application has online or offline access to the device parameter (see Clause 8).

Table 2 – List of defined root menu identifiers for PC-based devices

MENU identifier	Default STYLE	Usage	Short description
device_root_menu	MENU	Online	Device feature views for set up
diagnostic_root_menu	MENU	Online	Diagnostic views
maintenance_root_menu	MENU	Online	Maintenance feature views
offline_root_menu	TABLE	Offline	Offline configuration
process_variables_root_menu	MENU	Online	Process variable views

For HART EDDs the EDD application shall display offline_root_menu with a default STYLE WINDOW if STYLE is not defined.

PROFIBUS and PROFINET allow in submenus to define different parameter access by defining ACCESS ONLINE or ACCESS OFFLINE.

The EDDs should contain the online root menus from Table 2. The online root menus are optional for PROFIBUS, PROFINET and FOUNDATION fieldbus. The offline root menu is optional for FOUNDATION fieldbus. HART EDDs shall have all root menus in Table 2.

4.3.2 Online Root Menus

4.3.2.1 General

If none of the online root menus exists then communication profile specific entry points shall be used (see Table 3).

Table 3 – Fall back alternatives for online root menus

Communication profile	Description of fall back alternatives
FOUNDATION fieldbus	Menus declared in the BLOCK_A MENU_ITEMS attribute, e.g. device_root_menu_aiblock. If block based menus are not defined, then a MENU of style TABLE shall be generated from the BLOCK_A PARAMETERS attribute.
HART	root_menu
PROFIBUS, PROFINET	Menu_Main_Specialist if exists or Menu_Main_Maintenance. These menus include online and offline sub-menus.

4.3.2.2 Diagnostic Root Menu

The diagnostic_root_menu includes views that show the device state, detailed diagnostic information and may include graphical views that show, for example, a valve signature.

4.3.2.3 Process variable Root Menu

The process_variable_root_menu includes views that show process measurements and set points with their quality and important information for process operators, for example, ranges.

4.3.2.4 Device Root Menu

The device_root_menu includes features for a device. These features can be split into process-related and device-specific features. This structure is not required if the number of features is too small for splitting. Submenus that represent any structuring are allowed on the device feature menu. In case of such submenus, the menus that are underneath can be split into process-related and device-specific features.

4.3.2.5 Maintenance Root Menu

The maintenance_root_menu should contain features for maintaining the device during the runtime phase and e.g. showing information about last maintenance inspection.

4.3.3 Offline Root Menu

The offline_root_menu is a menu hierarchy including e.g. data items and methods for offline configuration. It contains, in particular, all of the application-specific parameters of the device and may also contain important read-only and writeable variables. For more information about application-specific parameters, see 9.3. The menu can have offline methods, for example, configuration assistants.

If the offline root menu does not exist, the communication profile specific entry point shall be used (see Table 4).

Table 4 – Fall back alternatives for offline root menus

Communication profile	Description of fall back alternatives
FOUNDATION fieldbus	BLOCK_A PARAMETERS
HART	MENU upload_variables
PROFIBUS, PROFINET	Table_Main_Specialist if exists or Table_Main_Maintenance

4.3.4 Example of EDD menu structure

The EDD example in Figure 1 includes additional menus that can be added to an EDD for PC applications. These menus are additional to the existing menus for the applications. This example is specific for a device with an interface according to HART. Examples for devices with an interface according FOUNDATION fieldbus, PROFIBUS and PROFINET would be very similar.

```

MENU diagnostic_root_menu
{
    LABEL "Diagnostics";
    STYLE MENU;                                /* not required to define STYLE in this case, */
                                              /* because of default STYLE of the root menu */

    ITEMS
    {
        status_window,                        /* menu: style=window */
        self_test                             /* method */
    }
}

MENU status_window
{
    LABEL "Status";
    STYLE WINDOW;
    ITEMS
    {
        standard_diagnostics_page,            /* menu: style=page */
        devspec_diagnostics_page              /* menu: style=page */
    }
}

MENU standard_diagnostics_page
{
    LABEL "Standard";
    STYLE PAGE;
    ITEMS
    {
        device_status                         /* variable */
    }
}

MENU devspec_diagnostics_page
{
    LABEL "Device Specific";
    STYLE PAGE;
    ITEMS
    {
        xmtr_specific_status_1,               /* variable */
        xmtr_specific_status_2                /* variable */
    }
}

METHOD self_test
{
    LABEL "Self Test";
    DEFINITION
    {
        /* elided */
    }
}

MENU maintenance_root_menu
{
    LABEL "Maintenance";
    STYLE MENU;                                /* not required to define STYLE in this case, */
                                              /* because of default STYLE of the root menu */

    ITEMS
    {
        device_mode_dialog,                   /* menu: style=dialog */
        teach_in                              /* method */
    }
}

MENU device_mode_dialog
{
    LABEL "Device Mode";
    STYLE DIALOG;
    ITEMS
    {
        mode_page                             /* menu: style=page */
    }
}

MENU mode_page
{
    LABEL "Process Variables";
    STYLE PAGE;
}

```

```

ITEMS
{
    transducer_group,      /* menu: style=group */
    function_group         /* menu: style=group */
}
}

MENU transducer_group
{
    LABEL "Transducer";
    STYLE GROUP;
    ITEMS
    {
        trans_target_mode, /* variable */
        trans_actual_mode  /* variable */
    }
}

MENU function_group
{
    LABEL "Function";
    STYLE GROUP;
    ITEMS
    {
        func_target_mode, /* variable */
        func_actual_mode  /* variable */
    }
}

METHOD teach_in
{
    LABEL "Teach-in";
    DEFINITION
    {
        /* elided */
    }
}

MENU process_variables_root_menu
{
    LABEL "Process Variables";
    STYLE MENU; /* not required to define STYLE in this case, */
               /* because of default STYLE of the root menu */

    ITEMS
    {
        overview_window, /* menu: style=window */
        primary_vars_window /* menu: style=window */
    }
}

MENU overview_window
{
    LABEL "Overview";
    STYLE WINDOW;
    ITEMS
    {
        process_vars_page /* menu: style=page */
    }
}

MENU process_vars_page
{
    LABEL "Process Variables";
    STYLE PAGE;
    ITEMS
    {
        pressure_group, /* menu: style=group */
        temperature_group /* menu: style=group */
    }
}

MENU pressure_group
{
    LABEL "Pressure";
    STYLE GROUP;
    ITEMS
    {
        pv_digital_value, /* variable */

```

```

        pv_upper_range_value,      /* variable */
        pv_lower_range_value      /* variable */
    }
}

MENU temperature_group
{
    LABEL "Temperature";
    STYLE GROUP;
    ITEMS
    {
        sv_digital_value,          /* variable */
        sv_upper_range_value,      /* variable */
        sv_lower_range_value      /* variable */
    }
}

MENU primary_vars_window
{
    LABEL "Primary Variables";
    STYLE WINDOW;
    ITEMS
    {
        pressure_chart_page,      /* menu: style=page */
        temperature_chart_page    /* menu: style=page */
    }
}

MENU pressure_chart_page
{
    LABEL "Pressure";
    STYLE PAGE;
    ITEMS
    {
        pressure_chart            /* chart */
    }
}

CHART pressure_chart
{
    /* elided */
}

MENU temperature_chart_page
{
    LABEL "Temperature";
    STYLE PAGE;
    ITEMS
    {
        temperature_chart        /* chart */
    }
}

CHART temperature_chart
{
    /* elided */
}

MENU device_root_menu
{
    LABEL "Device";
    STYLE MENU;
    /* not required to define STYLE in this case, */
    /* because of default STYLE of the root menu */

    ITEMS
    {
        process_related_window,   /* menu: style=window */
        device_specific_window,   /* menu: style=window */
        master_reset              /* method */
    }
}

MENU process_related_window
{
    LABEL "Process Related";
    STYLE WINDOW;
    ITEMS
    {
        identification_page,      /* menu: style=page */
    }
}

```

```

        output_info_page          /* menu: style=page */
    }
}

MENU identification_page
{
    LABEL "Identification";
    STYLE PAGE;
    ITEMS
    {
        tag,                      /* variable */
        manufacturer,             /* variable */
        device_type,              /* variable */
        device_revision,          /* variable */
        descriptor,               /* variable */
        message                    /* variable */
    }
}

MENU output_info_page
{
    LABEL "Output Information";
    STYLE PAGE;
    ITEMS
    {
        range_values_group,       /* menu: style=group */
        sensor_limits_group      /* menu: style=group */
    }
}

MENU range_values_group
{
    LABEL "Range Values";
    STYLE GROUP;
    ITEMS
    {
        pv_units,                /* variable */
        pv_urv,                  /* variable */
        pv_lrv                   /* variable */
    }
}

MENU sensor_limits_group
{
    LABEL "Sensor Limits";
    STYLE GROUP;
    ITEMS
    {
        sensor_units,            /* variable */
        upper_sensor_limit,      /* variable */
        lower_sensor_limit       /* variable */
    }
}

MENU device_specific_window
{
    LABEL "Device Specific";
    STYLE WINDOW;
    ITEMS
    {
        identification_page,     /* menu: style=page */
        calibration_page         /* menu: style=page */
    }
}

MENU calibration_page
{
    LABEL "Calibration";
    STYLE PAGE;
    ITEMS
    {
        sensor_limits_group,     /* menu: style=group */
        sensor_trim_group        /* menu: style=group */
    }
}

MENU sensor_trim_group
{

```

```

LABEL "Sensor Trim";
STYLE GROUP;
ITEMS
{
    upper_sensor_trim_point, /* variable */
    lower_sensor_trim_point, /* variable */
    sensor_trim              /* method */
}

METHOD master_reset
{
    LABEL "Master Reset";
    DEFINITION
    {
        /* elided */
    }
}

```

Figure 1 – EDD example of root menus

4.3.5 User interface

4.3.5.1 Diagnostics

Figure 2 shows an example of an EDD application for diagnostics.

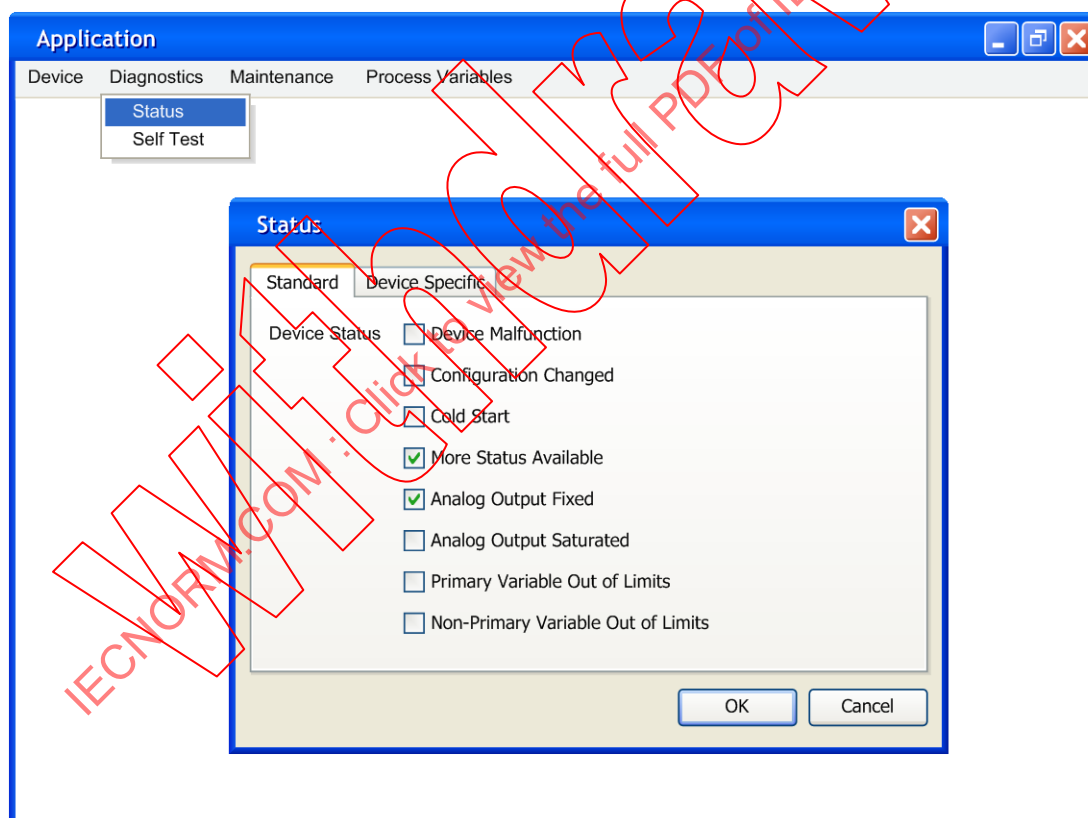


Figure 2 – Example of an EDD application for diagnostics

4.3.5.2 Process variables

Figure 3 and Figure 4 show examples of EDD applications for process variables.

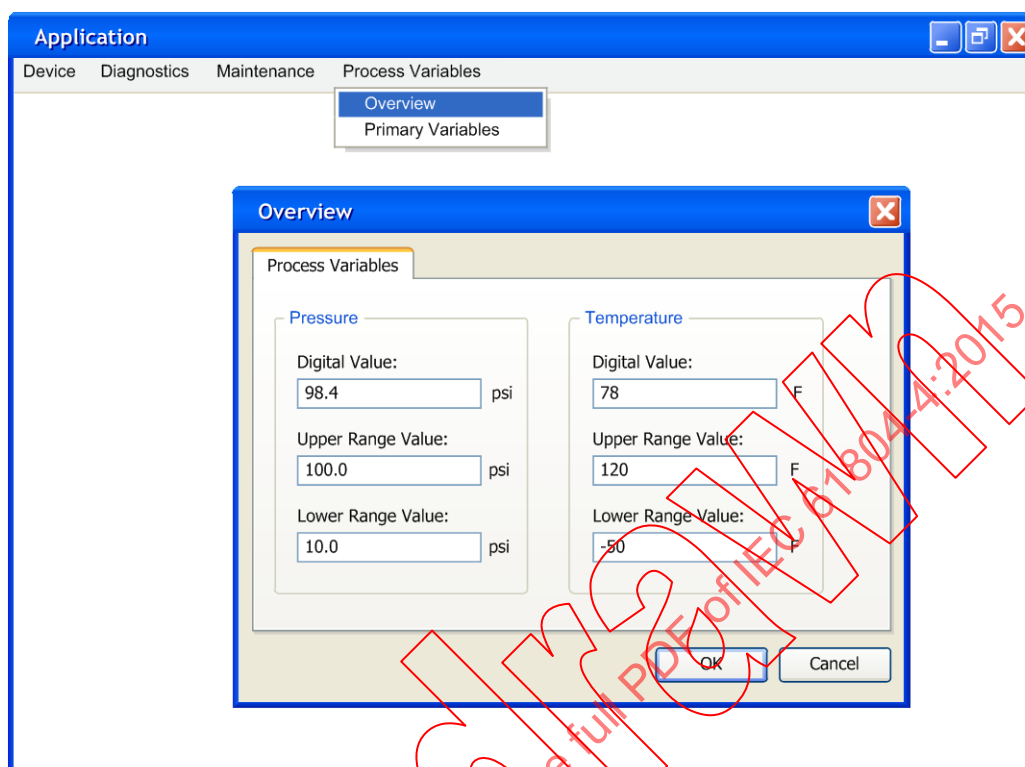


Figure 3 – Example of an EDD application for process variables

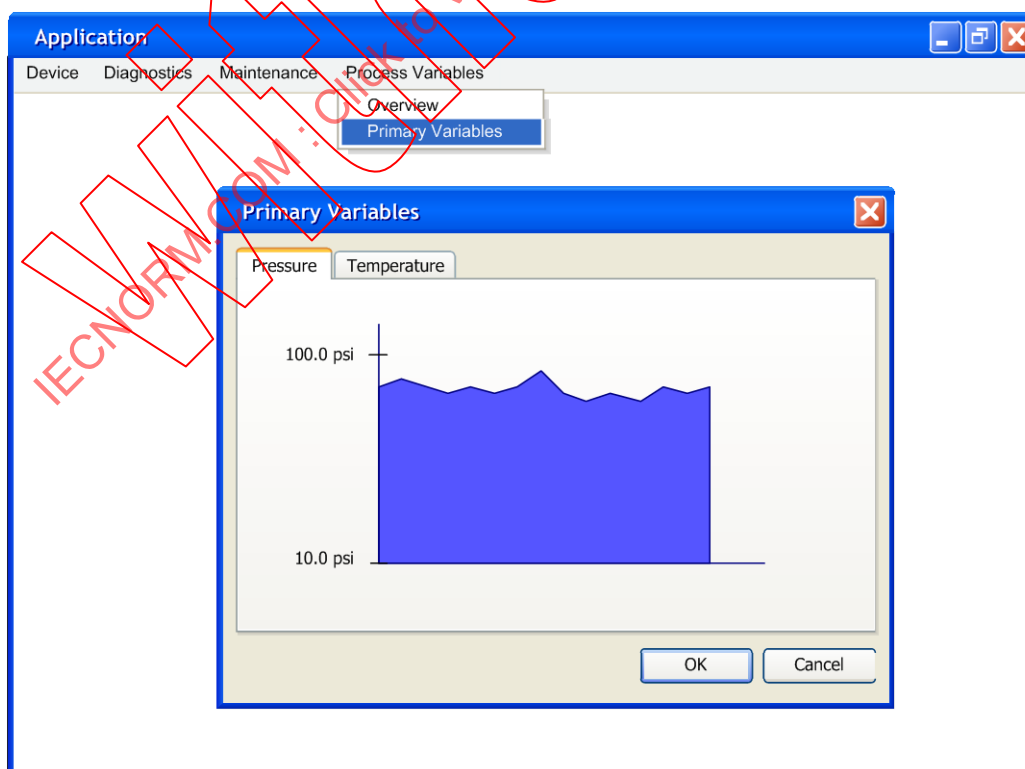


Figure 4 – Example of an EDD application for primary variables

4.3.5.3 Device features

Figure 5, Figure 6 and Figure 7 show examples of EDD applications for device features.

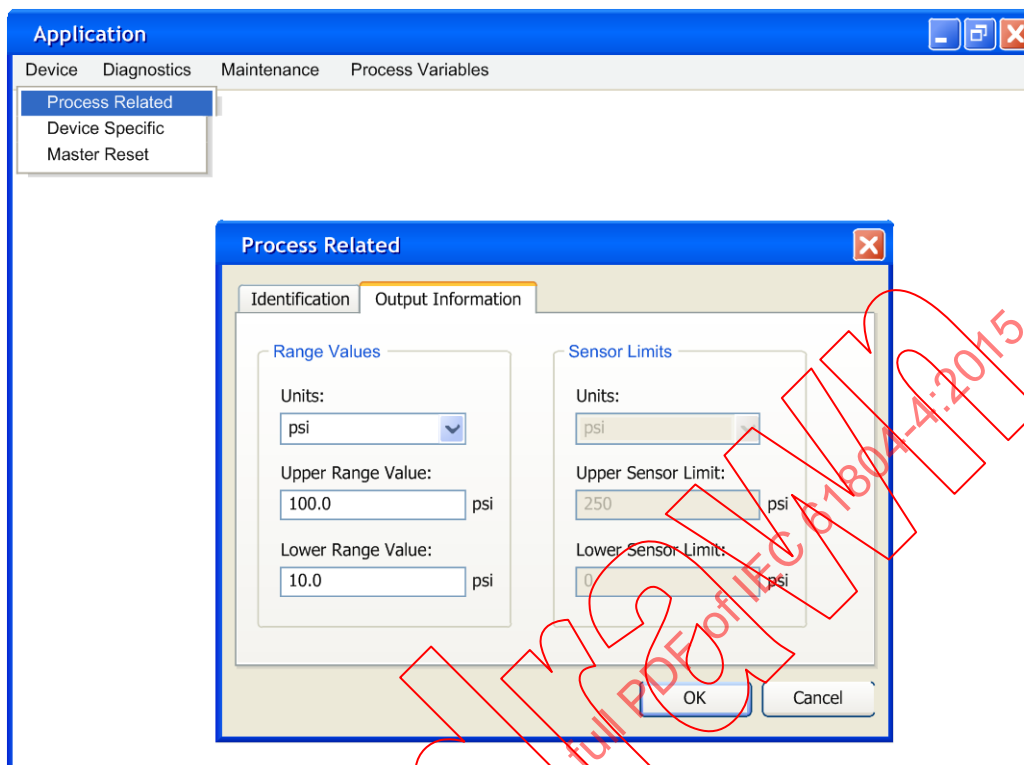


Figure 5 – Example of an EDD application for process-related device features

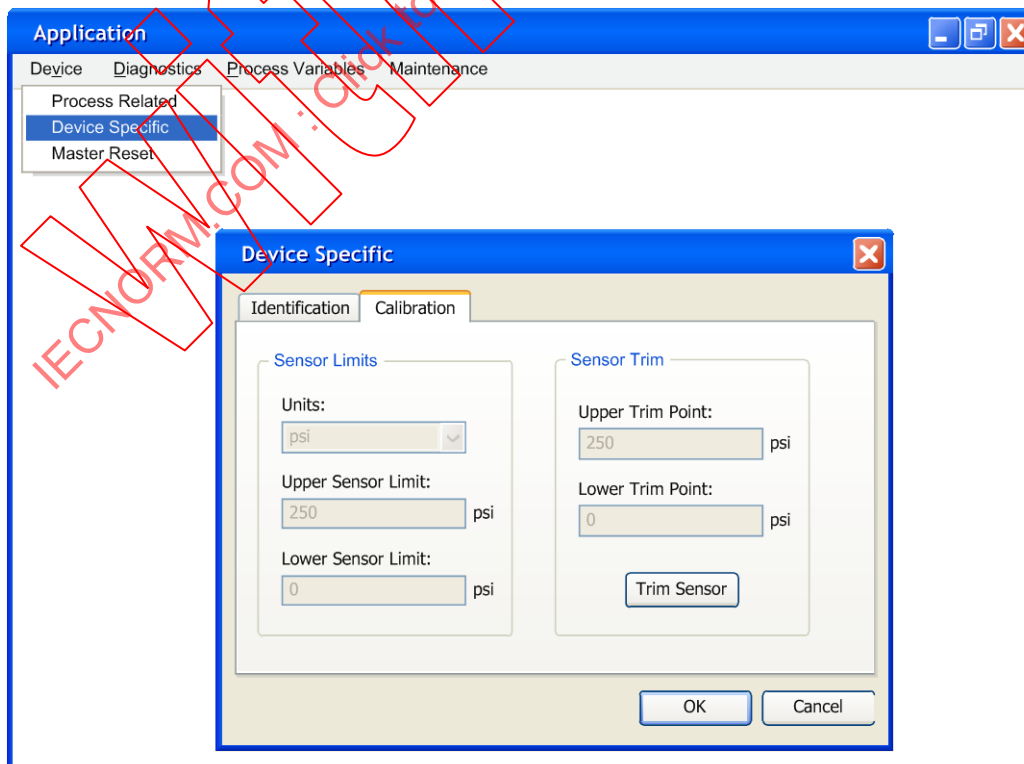


Figure 6 – Example of an EDD application for device features

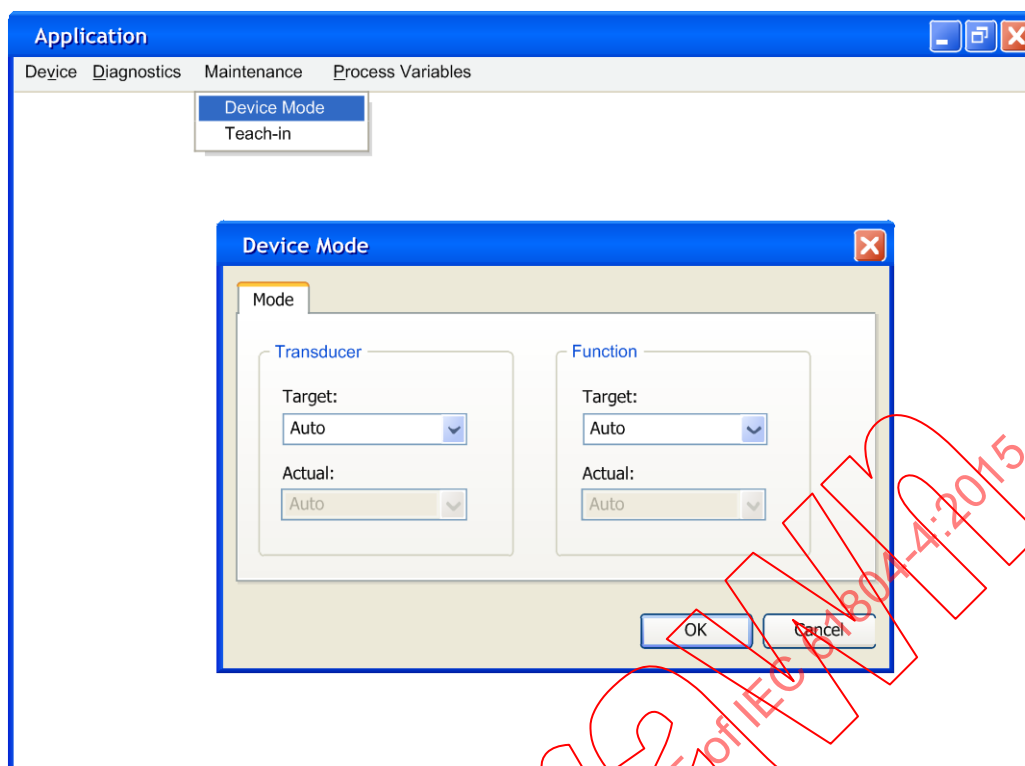


Figure 7 – Example of an EDD application for maintenance features

4.4 Containers and contained items

4.4.1 Overview

The user interface extensions are based on a simple user interface model. The model consists of two concepts:

- containers; and
- contained items.

Containers are so named because they contain other user interface elements. Containers may include: menus, windows, dialogs, tables, pages, groups, and other containers. Containers correspond to a MENU. They are distinguished from one another via the STYLE attribute. This STYLE attribute indicates how the MENU will be displayed.

Contained items include e.g. variables, methods, edit displays, graphs, charts, images, static text.

4.4.2 Containers

4.4.2.1 Permitted and default STYLES

Table 5 defines permitted user interface items in containers, substitutes and default STYLES are not defined. If the style of a menu is not defined, then a default style may be used depending on the menu where the menu is contained and the content of the menu.

The default style of a menu that is not contained in a menu has the style TABLE. The HART root_menu is shown as a parameter table.

General rules:

- If the ACCESS online or offline of a contained MENU is different to the access of the container, the EDD application shall use STYLE WINDOW in a window otherwise shall use STYLE DIALOG.

Table 5 – Permitted contained items and default STYLES

Container	Permitted contained items															Default STYLE of contained items, used if STYLE is not defined
	MENU	WINDOW, DIALOG	PAGE	GROUP	TABLE	EDIT_DISPL AY	VARIABLE	COLLECTIO N, RECORD	ARRAY, LIST	IMAGE, CHART, GRAPH, GRID	static text	METHOD	PLUGIN	COLUMNBR EAK, ROWBREAK	SEPARATO R	
MENU	X	X	D	D	2	X	7	2	2	2	2	X	X	6	6	MENU of STYLE MENU, if no data item is in the menu. DIALOG, if any data item is in the menu.
DIALOG, WINDOW	X	4	X	X	X	4	X	X	X	X	X	4	4	X	8	PAGE
PAGE	X	4	D	X	X	4	X	X	X	X	X	4	4	X	3	GROUP
GROUP	X	4	D	1	D	4	X	1	X	X	X	4	4	X	3	GROUP if the parent of the parent GROUP is not of STYLE GROUP. DIALOG rendered as button if the parent of the parent GROUP is of STYLE GROUP.
TABLE	4	D	D	D	X	4	X	X	X	5	X	4	4	6	6	TABLE
Key																
X Allowed.																
D Default STYLE shall be used (see column "Default STYLE of submenu if not defined").																
1 Only one sublevel is allowed e.g. GROUP may contain a GROUP but this GROUP shall not contain GROUPS COLLECTIONs or RECORDs																
2 Shall be rendered in a dialog.																
3 Shall be interpreted as COLUMNBREAK.																
4 Shall be rendered as button or hyperlink.																
5 Shall be rendered as button, hyperlink or inline.																
6 Shall be rendered as a line or ignored.																
7 Shall be rendered as button, hyperlink or ignored.																
8 SEPARATOR shall be treated as COLUMNBREAK.																

4.4.2.2 Menu

A menu of STYLE MENU takes the form of a drop-down menu, a pop-up menu or a navigation bar.

4.4.2.3 Window

A menu of STYLE WINDOW takes the form of a modeless window.

4.4.2.4 Dialog

A MENU of STYLE DIALOG shall be a modal window. If a dialog contains a subordinated window, the EDD application shall manage the window as a modal subdialog.

4.4.2.5 Table

A menu of STYLE TABLE shall be rendered as a table. Each item shall be represented as one or multiple rows, e.g. ARRAYS. The submenus build a hierarchy, which shall be shown.

The STYLE attribute (STYLE = TABLE) should be defined only on the root menu of a hierarchy.

4.4.2.6 Page

A menu of STYLE PAGE within a WINDOW or DIALOG shall be rendered as a tab.

The EDD application shall interpret ROWBREAKs or other items between pages as horizontal separations.

COLUMNBREAKs and SEPARATORs between pages shall be ignored.

4.4.2.7 Group

A menu of STYLE GROUP takes the form of a group box. In order to maintain a clear arrangement on the user interface, the following restricting rules apply.

- The nesting level of GROUP declarations is restricted to two. In other words, a GROUP may contain further GROUPs but these shall not contain further GROUPs. When a second layer GROUP contains a GROUP, this GROUP shall be rendered as either a button or a hyperlink. The LABEL shall appear on the button or as the hyperlink text. When the button or hyperlink is pressed, a corresponding dialog shall be opened on top of the calling dialog or window.

Within a GROUP, usually logically related parameters and methods are grouped. The title of the GROUP indicates this relation, for example, "AnalogInput 1". Often the same string is also used in the labels of the GROUP items, for example, "AnalogInput1_StaticRevision" or "Analog Input1_Blockmode". In order to avoid this duplication of information, the EDD developer may assemble the items of a GROUP in a COLLECTION in order to override the original labels. If the parameters are to be referred to in a GROUP the parameters can be referenced using the COLLECTION; in other cases, especially without additional semantic context, the parameters can be referenced directly.

Figure 8 shows an EDD example with COLLECTION MEMBERS within a MENU of STYLE GROUP.

```

VARIABLE ail_strev
{
    LABEL "Analog Input 1: Static Revision";
    TYPE INTEGER(4);
}

VARIABLE ail_bloMo
{
    LABEL "Analog Input 1: Block Mode";
    TYPE INTEGER(1);
}

MENU ail_group_double                                /* Leads to double display of */
{
    LABEL "Analog Input 1";
    STYLE GROUP;
    ITEMS
    {
        ail_strev,
        ail_bloMo
    }
}

COLLECTION ail_groupcol
{
    MEMBERS
    {
        strev,ail_strev,"Static Revision";
        bloMo,ail_bloMo,"Block Mode";
    }
}

MENU ail_group_single                                /* Leads to single display of */
{
    LABEL "Analog Input 1";
    STYLE GROUP;
    ITEMS
    {
        ail_groupcol.strev,
        ail_groupcol.bloMo
    }
}
    
```

Figure 8 – Usage of COLLECTION MEMBERS in MENUs of STYLE GROUP

4.4.3 Contained items

4.4.3.1 Overview

How a container renders contained items is described in 4.4.3. All containers render contained items in the same way. Contained items, such as methods, variables and others, have a LABEL attribute. In order not to disturb the layout of the EDD application, for example, with oversized buttons for methods, the EDD developer should not use very long strings for LABELs. For this reason, some EDD applications may truncate these labels.

4.4.3.2 Methods

A method should be rendered as a button or a hyperlink. The LABEL of the corresponding METHOD should appear on the button or as the hyperlink text. When the button or hyperlink is pressed, the corresponding METHOD should be executed.

4.4.3.3 Variables

4.4.3.3.1 General

The label, value and, if defined, the units of the variable should be displayed in a manner consistent with the definition of the corresponding VARIABLE.

The general handling of variables are as follows.

- HANDLING = READ or the menu item attribute is READ_ONLY: the value of the variable should not be editable.
- CLASS = DYNAMIC: the value should be updated continuously with current read values from the device.

4.4.3.3.2 Variable of TYPE BIT_ENUMERATED

Multiple bits can be packaged in a single-bit enumerated variable. Each bit could have a distinct meaning. It is possible to display each bit separately and, in addition, to display the whole BIT_ENUMERATED variable. BIT_ENUMERATED variables may be displayed as checkboxes.

In case of showing a bit of a BIT_ENUMERATED variable, the variable reference should be extended with a mask in square brackets and the LABEL of the variable is not shown.

Figure 9 shows an EDD example for displaying single bits of a BIT_ENUMERATED variable.

```
VARIABLE var1
{
    LABEL "Var 1";
    TYPE BIT_ENUMERATED
    {
        { 0x1, "Bit 0"},
        { 0x2, "Bit 1"},
        { 0x4, "Bit 2"}
    }
}

MENU diagnostic_info
{
    LABEL "Diagnostic";
    ITEMS
    {
        var1[0x1], // Bit 0
        var1[0x2], // Bit 1
        var1[0x4]  // Bit 2
    }
}
```

Figure 9 – Displaying single bits of BIT_ENUMERATED

Figure 10 shows an EDD example that shows the differences between the full display of a BIT_ENUMERATED variable and how it may be displayed if all bits are explicitly addressed.

```

VARIABLE var1
{
    LABEL "Var 1";
    TYPE BIT_ENUMERATED
    {
        { 0x1, "Bit 0"},
        { 0x2, "Bit 1"},
        { 0x4, "Bit 2"}
    }
}

MENU var1_group
{
    LABEL var1.LABEL;
    STYLE GROUP;
    ITEMS
    {
        var1[0x1],    // Bit 0
        var1[0x2],    // Bit 1
        var1[0x4]     // Bit 2
    }
}

MENU diagnostic_info
{
    LABEL "Diagnostic";
    STYLE PAGE;
    ITEMS
    {
        var1,          // all bit should be shown
        COLUMNBREAK,
        var1[0x1],      // Bit 0
        var1[0x2],      // Bit 1
        var1[0x4],      // Bit 2
        COLUMNBREAK,
        var1_group
    }
}

```

Figure 10 – Displaying multiple bits of BIT_ENUMERATED

Figure 11 shows how the result of the Figure 10 EDD example in an EDD application may appear.

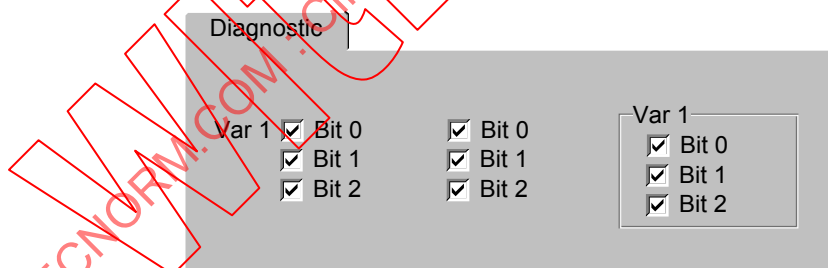


Figure 11 – Example of an EDD application for a variable of type BIT_ENUMERATED

4.4.3.3.3 Variable of TYPE INDEX

When a VARIABLE of type INDEX is presented to the user, the EDD application shall determine the text that shall be displayed by following rules:

- 1) if the description in the referenced ELEMENTS exist then this shall be displayed else;
- 2) if the LABEL of the variable exist then this shall be displayed else:
- 3) the LABEL of the array shall be displayed with the numeric value of the INDEX e.g. myarray[3].

If the INDEX value is out of range of the related array, a text should inform the user that the value is out of range.

If the variable is editable, it should be presented as a combo box.

4.4.3.4 Records and collections

RECORDs and COLLECTIONs shall be displayed as MENU with STYLE GROUP. COLLECTIONs can contain COLLECTIONs. This is handled like nested groups (see 4.4.2.7). The rules about nesting groups apply also in this case.

4.4.3.5 Arrays and lists

Value arrays, reference arrays and LISTs shall be rendered as a scrollable area within the container. The item LABEL shall always be visible.

NOTE HART does not support arrays and lists references in containers.

4.4.3.6 Edit displays

An EDIT_DISPLAY shall be rendered in the container as either a button or a hyperlink. The LABEL of the corresponding EDIT_DISPLAY shall appear on the button or as the hyperlink text.

When the button or hyperlink is activated, the corresponding EDIT_DISPLAY shall be opened as a modal window. The content of an EDIT_DISPLAY contains DISPLAY_ITEMS and EDIT_ITEMS. The EDD application shall display items from DISPLAY_ITEMS above of the items of EDIT_ITEMS. The DISPLAY_ITEMS shall be displayed as read only items and EDIT_ITEMS shall be handled like ITEMS from a MENU of STYLE DIALOG.

4.4.3.7 Graphs

A graph should be displayed in a manner that is consistent with the definition of the corresponding GRAPH. Refer to the layout rules defined in 4.5 and IEC 61804-3 for information on the affect of the WIDTH attribute on the layout of the GRAPH.

4.4.3.8 Charts

A chart should be displayed in a manner that is consistent with the definition of the corresponding CHART. Refer to the layout rules defined in 4.5 and IEC 61804-3 for information on the affect of the WIDTH attribute on the layout of the CHART.

4.4.3.9 Images

The images that are referenced by the MENU should be displayed.

The EDD application shall display an IMAGE in original size, if it is not referenced with a menu item INLINE qualifier. The EDD application shall insert a ROWBREAK above the IMAGE, if items exists before the IMAGE and insert a ROWBREAK below the IMAGE, if items exists after the IMAGE.

The EDD application shall scale down an IMAGE with a menu item INLINE qualifier if the IMAGE width is larger than the column width in which the IMAGE is contained.

The EDD application shall not scale up an IMAGE.

The EDD application shall keep original aspect ratio of IMAGES.

4.4.3.10 Static text

The text that is referenced by the MENU should be displayed. The text will be wrapped in multiple lines, if the text is longer than the width of dialog, window, page, group, or column.

4.4.3.11 Grid

The LABEL of the grid is shown at first and below the grid area with the width of the dialog, window, page, group, or column and the height that is needed to show the data or what is available on the screen. Scroll bars are used to scroll the grid area, if the width or height is too small to show the complete data.

4.5 Layout rules

4.5.1 Overview

Layout rules are rules that the EDD application shall follow for arranging the content of windows and dialogs on the display.

Each container defines the bounding box of the container. This is the area of the container where its contents shall be displayed. For example, the bounding box of a window or dialog is the entire window except the border and title bar.

The ITEMS of a container shall be displayed in the same order they appear in the EDD. The contents of the container should be organized starting at the upper left area of the container. The items shall be displayed vertically down the container. When a COLUMNBREAK is encountered, a new column shall be created. The following items shall be displayed in the next column starting at the top of the container. This process continues until all items have been displayed. Column breaks shall only be introduced when a COLUMNBREAK is encountered. The EDD application shall not introduce columns breaks on its own.

If static text contains carriage return and line feeds, then the text should be wrapped in lines if the text is long.

The ordering of items in unit and refresh relations cause or effect lists are not used for display purpose. If a WRITE_AS_ONE relation makes it necessary to complete the displayed items then these additional items shall appear in the order of the items in the WRITE_AS_ONE relation.

4.5.2 Layout rules for WIDTH and HEIGHT

VARIABLE, CHART, GRAPH and GRID have the attributes WIDTH and HEIGHT. The WIDTH attribute defines the number of columns of display. The HEIGHT attribute defines the vertical size of the display as a multiple of the minimum height for a standard input field, see Table 6. The default value of WIDTH and HEIGHT is MEDIUM for CHART, GRAPH, GRID; XXX_SMALL for VARIABLE height, and width.

The width of the EDD application should be split in up to a maximum of 5 columns depending on the number of COLUMNBREAKs and the WIDTH of the containers. Columns that would span to a column higher than the 5th column shall be moved to the next row through an implied ROWBREAK.

An implied ROWBREAK has the same behaviour as an explicitly defined ROWBREAK.

The WIDTH and HEIGHT attribute of a container defines how many columns the container spans and the height in multiples of minimum height possible for a simple field e.g. variable, constance string, menu and method reference, see Table 6.

Table 6 – WIDTH and HEIGHT span and applicability

WIDTH and HEIGHT Value	WIDTH	HEIGHT	Applicability
XXX_SMALL	1	1	VARIABLE
XX_SMALL	1	3	CHART GRAPH GRID VARIABLE
X_SMALL	1	5	CHART GRAPH GRID VARIABLE
SMALL	2	7	CHART GRAPH GRID VARIABLE
MEDIUM	3	8	CHART GRAPH GRID VARIABLE
LARGE	4	9	CHART GRAPH GRID VARIABLE
X_LARGE	5	11	CHART GRAPH GRID VARIABLE
XX_LARGE	5	13	CHART GRAPH GRID VARIABLE

4.5.3 Layout rules for COLUMNBREAK and ROWBREAK

4.5.3.1 Layout for protruding elements

The EDD application shall insert all the menu items in a single column one after the other until it encounters a COLUMNBREAK. The COLUMNBREAK will cause the next menu item to be inserted in the highest row in the next column. If there is a menu item occupying a region of space in the next column, then the menu item will be inserted in the row immediately below the occupying menu item. See Figure 12 for EDD source code example and Figure 13 for the corresponding layout.

```

MENU protruding_elements
{
    LABEL "protruding_elements";
    STYLE WINDOW;
    ITEMS
    {
        Element1,
        Element2,
        Element3,
        Element4,
        SmallGraph5,      //WIDTH SMALL (2 columns), HEIGHT SMALL (7 height units)
        Element6,
        Element7,
        Element8,
        COLUMNBREAK,
        Element9,
        Element10,
        Element11,
        COLUMNBREAK,
        Element12,
        Element13,
        Element14,
    }
}

```

Figure 12 – EDD source code for layout for protruding elements example

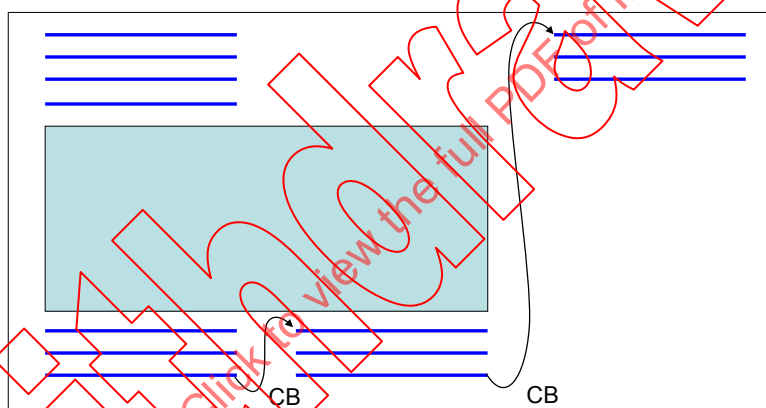


Figure 13 – Layout for protruding elements

4.5.3.2 Layout for partially filled rows

The EDD application shall insert all the menu items in a single column one after the other until it encounters a ROWBREAK. A ROWBREAK will cause the next menu item to be inserted in the lowest row in the 1st column (it acts as a carriage return). The ROWBREAK shall also produce a horizontal barrier so that any further COLUMNBREAK will prevent a menu item from being inserted above this line. See Figure 14 for EDD source code example and Figure 15 for the corresponding layout.

```

MENU Fig13
{
    LABEL "Fig13";
    STYLE WINDOW;
    ITEMS
    {
        Element1,
        Element2,
        Element3,
        COLUMNBREAK,
        Element4,
        Element5,
        Element6,
        Element7,
        ROWBREAK,
        SmallGraph8,    //WIDTH SMALL, HEIGHT SMALL, occupies 2 columns
        Element9,
        Element10,
        Element11,
        COLUMNBREAK,
        Element12,
        Element13,
        Element14,
        COLUMNBREAK,
        Element15,
        Element16,
        Element17,
    }
}

```

Figure 14 – EDD source code for layout for partially filled rows example

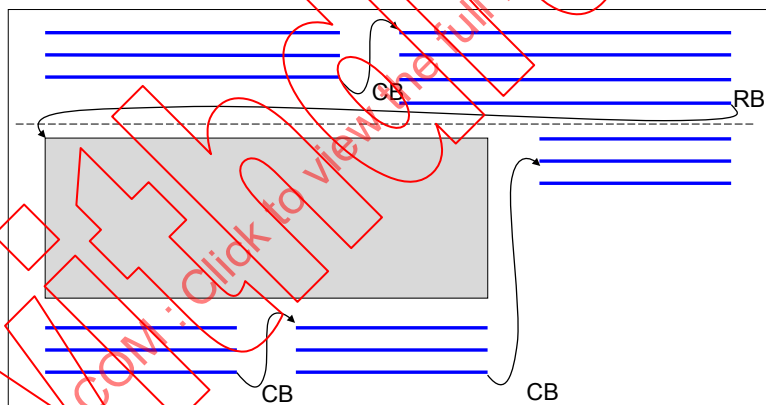


Figure 15 – Layout for partially filled rows

Figure 17 shows that menu items in the rows with two columns (rows 1 and 3) have expanded cell space available such that the row takes up the same width as the rows with three columns. Figure 16 shows the corresponding EDD source code example.

```

MENU partially_filled_rows
{
    LABEL "partially filled rows";
    STYLE WINDOW;
    ITEMS
    {
        Element1,
        Element2,
        COLUMNBREAK,
        Element3,
        Element4,
        ROWBREAK,
        Element5,
        Element6,
        COLUMNBREAK,
        Element7,
        Element8,
        COLUMNBREAK,
        Element9,
        Element10,
        ROWBREAK,
        Element11,
        Element12,
        COLUMNBREAK,
        Element13,
        Element14
    }
}
    
```

Figure 16 – EDD source code for layout for partially filled rows example

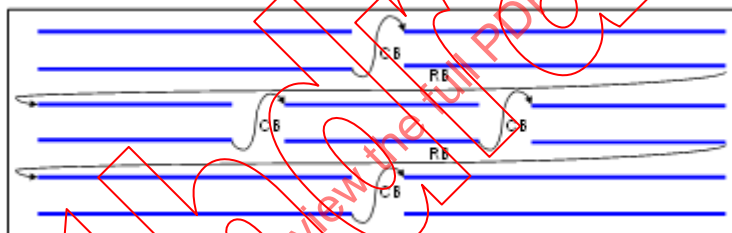


Figure 17 – Layout for partially filled rows

4.5.3.3 Layout for oversized elements

The EDD application shall insert all the menu items in a single column one after the other until it encounters a COLUMNBREAK. The COLUMNBREAK will cause the next menu item to be inserted in the highest row in the next column. If the menu item to be inserted is wider than the number of columns available, then the menu item will be inserted in the lowest row in the 1st column (i.e. implied ROWBREAK). See Figure 18 for EDD source code example and Figure 19 for the corresponding layout.

```

MENU layout_for_oversized_elements
{
    LABEL "layout for oversized elements";
    STYLE WINDOW;
    ITEMS
    {
        Element1,
        Element2,
        Element3,
        COLUMNBREAK,
        Element4,
        Element5,
        Element6,
        COLUMNBREAK,
        Element7,
        Element8,
        Element9,
        COLUMNBREAK, //Results in implied ROWBREAK
        MediumGraph10 //WIDTH MEDIUM, HEIGHT MEDIUM, occupies 3 columns
    }
}

```

Figure 18 – EDD source code for layout for oversized elements example

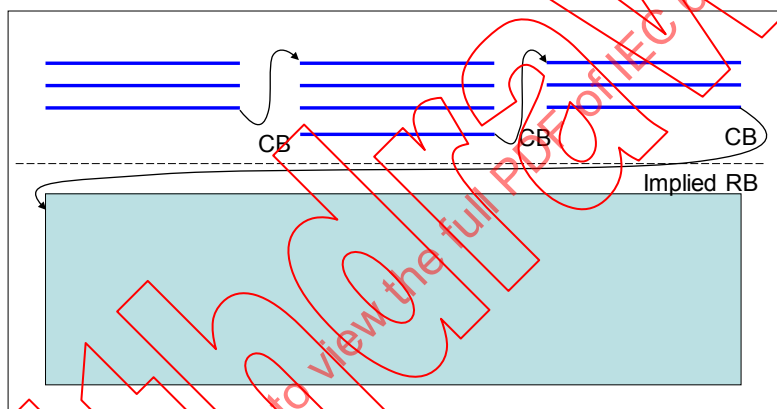


Figure 19 – Layout for oversized elements

4.5.3.4 Layout for columns in stacked group

The EDD application shall insert all the menu items in a single column one after the other until it encounters a COLUMNBREAK. The COLUMNBREAK will cause the next menu item to be inserted in the highest row in the next column. If the menu item resides in a nested menu (such as a GROUP), then the item will remain inside the parent menu. See Figure 20 for EDD source code example and Figure 21 for the corresponding layout.

```

MENU layout_for_columns_in_stacked_group
{
    LABEL "layout for columns in stacked group";
    STYLE WINDOW;
    ITEMS
    {
        Element1,
        Element2,
        Element3,
        Element4,
        GroupWithThreeColumns5,
        COLUMNBREAK,
        COLUMNBREAK,
        COLUMNBREAK,
        Element6,
        Element7,
        Element8,
        Element9,
        Element10,
        Element11,
        Element12,
        Element13
    }
}
    
```

Figure 20 – EDD source code example for a layout for columns in stacked group

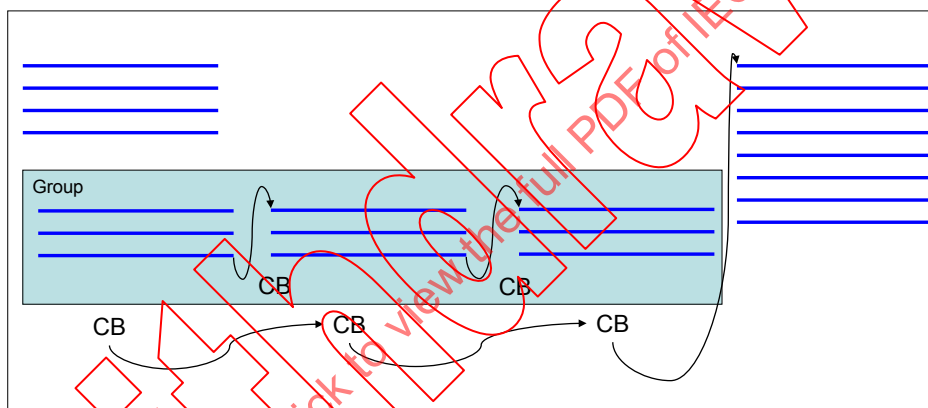


Figure 21 – Layout for columns in stacked group

Graphs inside groups shall be handled equivalently as multiple columns inside nested group. See Figure 22 for EDD source code example and Figure 23 for the corresponding layout.

```

MENU layout_for_columns_in_stacked_group_with_a_graph
{
    LABEL "layout for columns in stacked group with a graph";
    STYLE WINDOW;
    ITEMS
    {
        Element1,
        Element2,
        Element3,
        Element4,
        GroupWithTreeEllementsOneColumnsBreakChartWidthSmall,
        COLUMNBREAK,
        COLUMNBREAK,
        COLUMNBREAK,
        Element8,
        Element9,
        Element10,
        Element11,
        Element12,
        Element13,
        Element14,
        Element15
    }
}

```

Figure 22 – EDD source code for layout for columns with GRAPHS in stacked group example

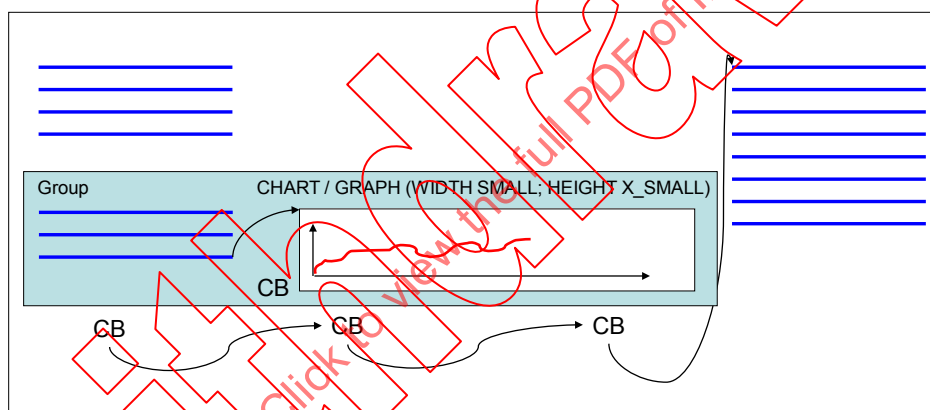


Figure 23 – Layout for columns with GRAPHS in stacked group

4.5.4 Layout examples

4.5.4.1 Single-column layout

The simplest layout is a list of variables, methods, edit displays, static text, graphs and charts.

```

MENU overview_window
{
    LABEL "Overview";
    STYLE WINDOW;
    ITEMS
    {
        primary_variable,           /* variable */
        upper_range_value,          /* variable */
        lower_range_value,          /* variable */
        master_reset,               /* method */
        self_test                   /* method */
    }
}

```

Figure 24 – Example of an EDD for an overview menu

The EDD in Figure 24 displays the items vertically in a window. The items are displayed starting at the left side of the screen and take up as much of the window as needed, see Figure 25.

Figure 25 – Example of an EDD application for an overview window

4.5.4.2 Multi-column and -row layout

Multiple columns of variables, methods, groups, images, graphs and charts may also be used, see Figure 26.

```
MENU overview_page
{
  LABEL "Overview";
  STYLE PAGE;
  ITEMS
  {
    upper_range_value, /* variable */
    lower_range_value, /* variable */
    upper_sensor_limit, /* variable */
    lower_sensor_limit, /* variable */
    COLUMNBREAK, /* column break */
    tag, /* variable */
    units, /* variable */
    damping, /* variable */
    transfer_function /* variable */
  }
}
```

Figure 26 – Example of an EDD using COLUMNBREAK

A COLUMNBREAK in Figure 26 is used to indicate a column break. All of the elements preceding the COLUMNBREAK will be placed in one column and all the elements following the COLUMNBREAK will be placed into the next column, see Figure 27.

Upper Range Value:	20	psi	Tag:	PT-101
Lower Range Value:	10	psi	Units:	psi
Upper Sensor Limit:	100	psi	Damping:	15 sec
Lower Sensor Limit:	0	psi	Transfer Function:	Linear

Figure 27 – Example of an EDD application for an overview window

A ROWBREAK in Figure 28 is used to place the following menu items beginning on the left side, see Figure 29.

```

MENU overview_page
{
  LABEL "Overview";
  STYLE PAGE;
  ITEMS
  {
    tag,                      /* variable */
    ROWBREAK,
    range_values,             /* group */
    COLUMNBREAK,
    sensor_limits,           /* group */
    ROWBREAK,
    units,                   /* variable */
    damping,                 /* variable */
    transfer_function         /* variable */
    COLUMNBREAK
  }
}

MENU range_values
{
  LABEL "Range Values";
  STYLE GROUP;
  ITEMS
  {
    upper_range_value,        /* variable */
    lower_range_value,        /* variable */
  }
}

MENU sensor_limits
{
  LABEL "Sensor Limits";
  STYLE GROUP;
  ITEMS
  {
    upper_sensor_limit,       /* variable */
    lower_sensor_limit,       /* variable */
  }
}

```

Figure 28 – EDD example for an overview window

Figure 29 shows the result of Figure 28 in an EDD application. Blue dashed boxes show the available space.

Figure 29 – Example of an EDD application for an overview window

4.5.4.3 In-line graphs and charts

Graphs and charts may appear within the columns just like variables and methods, see Figure 30 and Figure 31.

```
MENU overview_page
{
  LABEL "Overview";
  STYLE PAGE;
  ITEMS
  {
    primary_variable,      /* variable */
    upper_range_value,     /* variable */
    lower_range_value,     /* variable */
    COLUMNBREAK,          /* column break */
    pv_graph               /* graph */
  }
}
```

Figure 30 – Example of an EDD for in-line graphs and charts

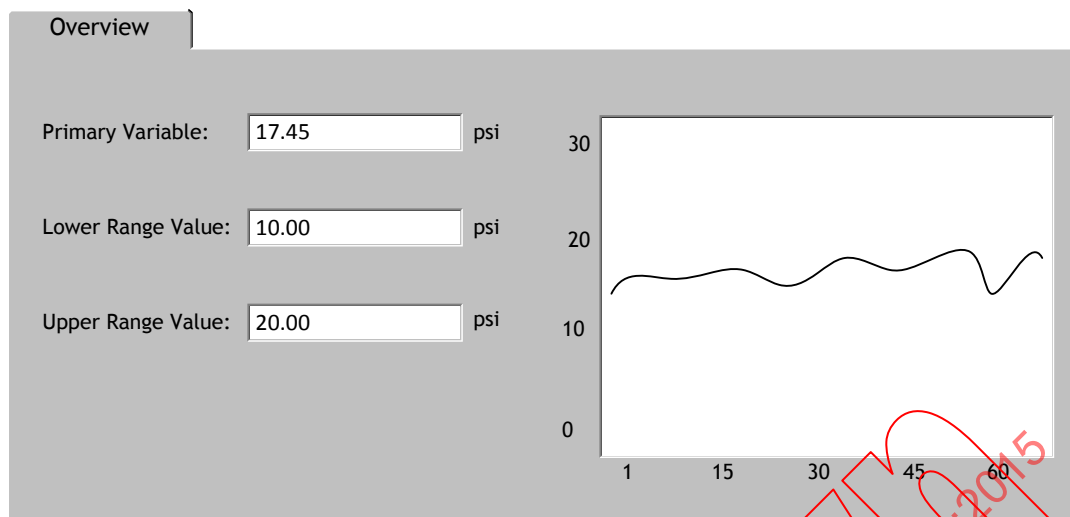


Figure 31 – Example of an EDD application for an in-line graph

If a graph or chart has with a WIDTH attribute that is equal to XX_SMALL, X_SMALL, SMALL or MEDIUM, it shall be displayed within the column. When the WIDTH equals LARGE, X_LARGE, or XX_LARGE the description in 4.5.4.4 shall apply.

4.5.4.4 Full-width graphs and charts

Graphs and charts may also span multiple columns, see Figure 32 and Figure 33.

```
GRAPH pv_graph
{
    WIDTH          MEDIUM;
    HEIGHT         MEDIUM;
    ...
}

MENU overview_page
{
    LABEL "Overview";
    STYLE PAGE;
    ITEMS
    {
        primary_variable, /* variable */
        COLUMNBREAK,      /* column break */
        upper_range_value, /* variable */
        COLUMNBREAK,      /* column break */
        lower_range_value, /* variable */
        pv_graph,          /* graph */
        reload_pv_graph,   /* method */
        COLUMNBREAK,      /* column break */
        save_pv_graph      /* method */
    }
}
```

Figure 32 – Example of an EDD for full-width graphs and charts

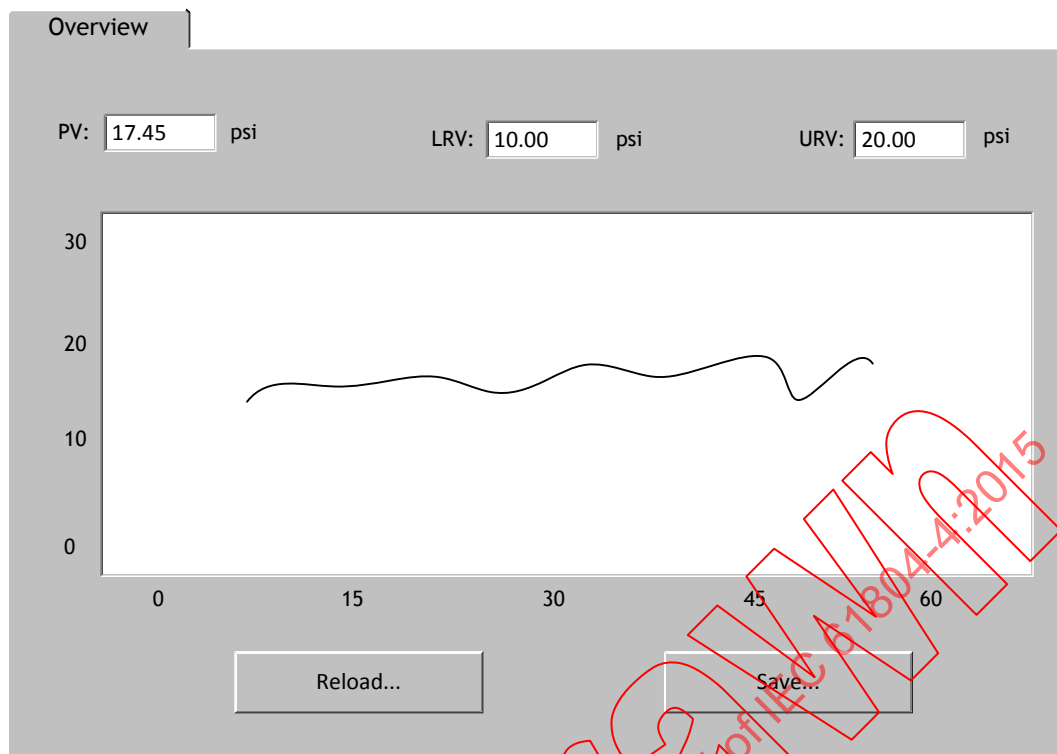


Figure 33 – Example of an EDD application for a full-width graph

If a graph or chart has a WIDTH attribute that is equal to X_LARGE, or XX_LARGE, it will span the width of the window, dialog, page or group. There may be additional elements before and after the graph, in which case the elements before the graph or chart are broken into one set of columns in a separate row and the elements following the graph or chart are broken into another set of columns in a separate row.

NOTE There are three columns prior to the graph and two columns following the graph.

4.5.4.5 Nested containers

Containers can be nested within other containers. Figure 34 and Figure 35 show an example of a page that is nested within a window and of two groups that are nested within the page.

```

MENU overview_window
{
    LABEL "Device Overview";
    STYLE WINDOW;
    ITEMS
    {
        overview_page                                /* page */
    }
}
MENU overview_page
{
    LABEL "Overview";
    STYLE PAGE;
    ITEMS
    {
        range_values,                                /* group */
        sensor_limits,                               /* group */
        COLUMNBREAK,                                /* column break */
        tag,                                          /* variable */
        units,                                       /* variable */
        damping,                                    /* variable */
        transfer_function                            /* variable */
    }
}
MENU range_values
{
    LABEL "Range Values";
    STYLE GROUP;
    ITEMS
    {
        upper_range_value,                          /* variable */
        lower_range_value                           /* variable */
    }
}
MENU sensor_limits
{
    LABEL "Sensor Limits";
    STYLE GROUP;
    ITEMS
    {
        upper_sensor_limit,                          /* variable */
        lower_sensor_limit                           /* variable */
    }
}

```

Figure 34 – Example of an EDD for nested containers

The screenshot shows a graphical user interface titled "Device Overview". It features a nested container structure. The main container has a tab labeled "Overview". Inside this container, there are two sub-containers: "Range Values" and "Sensor Limits". The "Range Values" container contains two input fields: "Upper Range Value" with a value of 20 and "Lower Range Value" with a value of 10, both followed by "psi" units. The "Sensor Limits" container contains two input fields: "Upper Sensor Limit" with a value of 100 and "Lower Sensor Limit" with a value of 0, both followed by "psi" units. To the right of these sub-containers, there are four input fields: "Tag" with the value "PT-101", "Units" with a dropdown menu showing "psi", "Damping" with a value of 15 and "sec" units, and "Transfer Function" with a dropdown menu showing "Linear".

Figure 35 – Example of an EDD application for nested containers

4.5.4.6 Edit displays

EDIT_DISPLAYs can be referenced in containers. When an edit display appears in a container, it shall be represented as a button or hyperlink. When the button is activated, a dialog that displays the contents of the edit display shall appear, see Figure 36 and Figure 37. The OK and Cancel buttons are not in scope of this specification.

```
MENU overview_window
{
    LABEL "Device Overview";
    STYLE WINDOW;
    ITEMS
    {
        overview_page                                /* page */
    }
}
MENU overview_page
{
    LABEL "Overview";
    STYLE PAGE;
    ITEMS
    {
        tag,                                           /* variable */
        units,                                         /* variable */
        damping,                                       /* variable */
        transfer_function,                             /* variable */
        range_values                                   /* edit display */
    }
}
EDIT_DISPLAY range_values
{
    LABEL "Range Values";
    DISPLAY_ITEMS
    {
        upper_sensor_limit,                           /* variable */
        lower_sensor_limit                             /* variable */
    }
    EDIT_ITEMS
    {
        upper_range_value,                             /* variable */
        lower_range_value,                             /* variable */
        units                                           /* variable */
    }
}
```

Figure 36 – Example of an EDD for EDIT_DISPLAYs

The figure illustrates the application of an EDD (Edit Display Dialog) for EDIT_DISPLAYs. It shows a 'Device Overview' window with a 'Range Values...' button. A modal dialog titled 'Range Values' is open, displaying fields for 'Upper Sensor Limit' (100 psi), 'Lower Sensor Limit' (0 psi), 'Upper Range Value' (20 psi), 'Lower Range Value' (10 psi), and 'Units' (psi). The dialog has 'OK' and 'Cancel' buttons.

Figure 37 – Example of an EDD application for EDIT_DISPLAYs

4.5.4.7 Images

Images can also be placed in containers. If an image is referenced with a menu item **INLINE** qualifier, the image is displayed within the current column of the container, see Figure 38 and Figure 39. Otherwise, the image is displayed across the width of the container.

```

MENU overview_page
{
  LABEL "Overview";
  STYLE PAGE;
  ITEMS
  {
    voltage,           /* Reference to an image */
    ramp_start,        /* variable */
    ramp_end,          /* variable */
    COLUMNBREAK,
    logo(INLINE)       /* Reference to an image which is displayed inline */
  }
}

```

Figure 38 – Example of an EDD for images

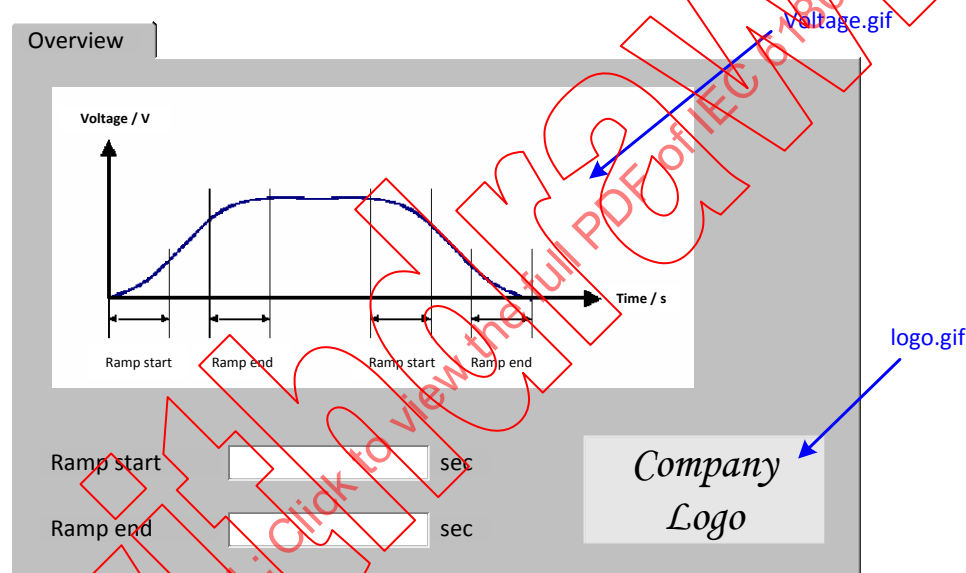


Figure 39 – Example of an EDD application for images

4.5.5 Conditional user interface

4.5.5.1 Overview

To control the appearance of user interface elements, the EDDL has different possibilities:

- the attribute **VISIBILITY**;
- conditional expressions on the **MENU ITEM**;
- the attribute **VALIDITY**.

4.5.5.2 VISIBILITY

The attribute **VISIBILITY** of the basic constructs (e.g. **MENU**, **VARIABLE**, **IMAGE**, **GRAPH**, **CHART**, **GRID**) allows controlling the appearance of user interface elements. The values **TRUE** or **FALSE** of **VISIBILITY** are meant to be evaluated dynamically depending on **VARIABLE** values.

In case of a **MENU**, **VISIBILITY** of the child **MENU** does not affect the screen layout of the **MENU**. In the screen layout of the **MENU**, the same space is reserved as if the child **MENU** is shown.

4.5.5.3 Conditional expressions in MENU ITEM

Conditional expressions in the MENU ITEM list allow controlling the screen layout. In the screen layout, no space is reserved if an element is not shown.

4.5.5.4 VALIDITY

The attribute VALIDITY of the basic constructs (e.g. MENU, VARIABLE, IMAGE, GRAPH, CHART, GRID) allows controlling the appearance of user interface elements. The values TRUE or FALSE of VALIDITY are meant to be evaluated dynamically depending on VARIABLE values.

VALIDITY does affect the screen layout. In the screen layout, no space is reserved if the element is not valid.

NOTE Communication is influenced by VALIDITY, see Clause 10.

4.6 Graphical elements

4.6.1 Overview

Smart microprocessor-based field devices continue to get more sophisticated and complex. In some cases, measurements or controllers that used to be impractical or required many pieces of equipment have been incorporated into a single device. EDDL supports these very sophisticated devices.

Of course, these constructs can be used in many other types of devices. The EDD developer has complete control over the content of the EDD and ultimately decides what EDDL constructs are to be used.

To address the needs of these complex devices, support for the following capabilities was added:

- graphing;
- charting;
- pictorial content;
- tabular data.

EDDL supports two kinds of graphical data visualizations. These are GRAPH and CHART. A CHART is used to display actual values and a GRAPH is used to display stored data that may be read from the device or persistent storage. One of the common uses is to compare a waveform from the device to a reference waveform or waveform from a specific date/operation, which is stored by the EDD application. The main difference between both constructs is that the EDD application handles the data of a CHART, and for a GRAPH the EDD itself defines, reads and updates the data. Methods for data handling are optional for CHARTs but they are typically needed for GRAPHS.

A GRAPH contains a list of WAVEFORMs that are plotted together. The GRAPH may be referenced from a MENU or from within a METHOD. WAVEFORMs have a minimal number of mandatory attributes in order to allow a graph to be easily produced by the EDD developer. For those wanting more control, a wide range of optional attributes is available (display size, axis, key points, etc.).

These constructs are particularly useful for plotting valve signatures and radar signals. Both data from the field device and data stored by the EDDL application may be plotted on the same GRAPH. For example, data from the field device can be specified in one WAVEFORM and persistent data from a FILE in another WAVEFORM. The two WAVEFORMs can be plotted on the same GRAPH.

While a GRAPH is a snapshot of the device or process properties, a CHART provides a view of a time varying trend. A CHART has SOURCES that are sampled periodically. STRIP, Sweep and SCOPE style plots are supported so as to show trends over time. Horizontal, vertical, and gauge types allow graphical presentation of a single instantaneous value.

For some devices, the condition and performance cannot be determined empirically. For these devices, the relative performance is the indicator of the condition of the device. The FILE construct allows access to device data stored by the EDDL application. The EDD specifies the data, which is to be stored with a similar syntax to COLLECTION. The EDD developer defines the data to be stored in any combination of VARIABLES, ARRAYS, COLLECTIONs, or LISTs.

The LIST construct was added to improve language flexibility when specifying persistent data stored in a FILE. The LIST is a variable length array. Data can be added to or removed from the list over time. Each entry in the LIST is of the same type as specified by the EDD developer.

The COLLECTION construct supports any combination of member types. Previously, collections were only allowed to contain references of one type (VARIABLE, ARRAY, COLLECTION, MENU).

Subclause 4.6 provides guidance and expectations for the use and support of EDDL constructs for graphical representations.

4.6.2 Graph and chart

4.6.3 Common attributes

4.6.3.1 Sizes

The HEIGHT and WIDTH attributes define the size of the CHART and GRAPH relative to the window rendered by the EDD application. HEIGHT and WIDTH do not provide absolute values for the graph window, but rather provide a range of values from XX_SMALL to XX_LARGE. The EDD application is free to establish the actual graph window size. The EDD application should render CHARTs and GRAPHs that refer to the same HEIGHT or WIDTH value of the same proportion. The default setting for HEIGHT and WIDTH is MEDIUM, see Figure 40. The EDD application should render CHARTs and GRAPHs that have the same HEIGHT or WIDTH with the same proportion.

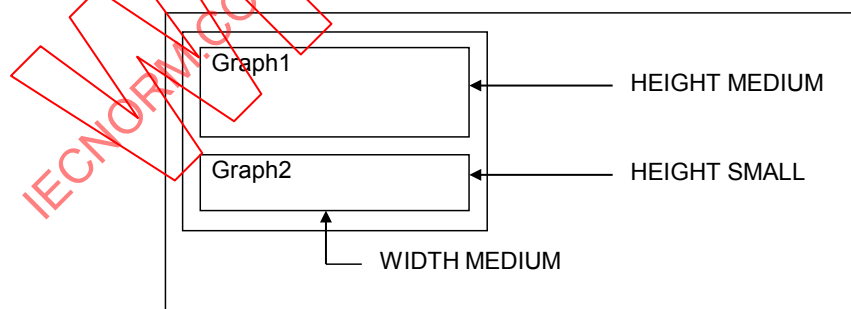


Figure 40 – HEIGHT and WIDTH attributes for CHART and GRAPH

4.6.3.2 Line characteristics

The LINE_TYPE attribute defines categories for display attributes for WAVEFORMs that are rendered on GRAPHs and SOURCEs that are rendered on CHARTs. Such display attributes may include colour and line styles (dash, dotted, etc.). The EDD application may provide user-configurable attributes for each LINE_TYPE. WAVEFORMs and SOURCEs with the same LINE_TYPE attribute should be rendered with the characteristics. Categories exist for data that are classified with DATA 1 to DATA 9, limits and TRANSPARENT, which only provides the display with the values and does not provide the connecting line.

The EMPHASIS attribute is used to differentiate between one or more SOURCES or WAVEFORMs on a given CHART or GRAPH. By default, the WAVEFORM or SOURCE with the EMPHASIS attribute that is set to TRUE should be displayed with a larger weight (see Figure 41). The EDD application may provide user configurable attributes for EMPHASIS.

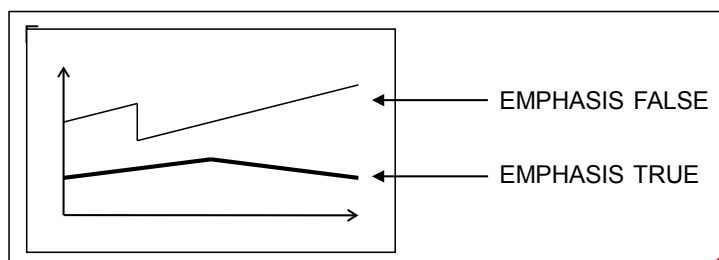


Figure 41 – EMPHASIS attribute to differentiate one or more SOURCES or WAVEFORMs

4.6.4 CHART

4.6.4.1 Overview

A CHART is used to display continuous data values, as opposed to a GRAPH, which is used to display a dataset. The chart supports multiple data sources and multiple Y-axes. Methods can be defined for data retrieval, scrolling and zooming support.

The background colour of a CHART display is determined by the EDD application. To provide a common look and feel, it is recommended that the background colour be white. It is recommended that an EDD application support at least six curves to be displayed simultaneously.

The elements of CHART are SOURCE (this defines the source of data), visualization elements, chart type and actions.

4.6.4.2 Chart types

4.6.4.2.1 Overview

A chart can be shown in different ways. It can be shown as a horizontal or vertical bar chart, as a chart with continuously updated waveforms or as a gauge. If the type is not defined, the default is STRIP.

4.6.4.2.2 GAUGE

The actual visual element of a gauge is decided by the EDD application. A CHART with a type GAUGE can have only one data source.

4.6.4.2.3 HORIZONTAL_BAR

The actual format (visual elements) of the display is decided by the EDD application. This type dictates a horizontal bar graph type of display. This type of display can have multiple bar charts (SOURCE). A bar chart is displayed for each variable.

4.6.4.2.4 SCOPE

In SCOPE, when the source values reach the end of the display area of the CHART, the display area is erased. The new source values are once again displayed, starting at the beginning of the display area. This type of display can have multiple SOURCE definitions.

4.6.4.2.5 STRIP

In STRIP, when the source values reach the end of the display area of the CHART, the display area is scrolled. The oldest source values are then removed from the display area and the newest source values are added. This type of display can have multiple SOURCE definitions.

4.6.4.2.6 SWEEP

In SWEEP, when the source values reach the end of the display area of the CHART, the new source values are once again displayed, starting at the beginning of the display area. Unlike SCOPE, only the portion of the display area needed to display the new source values is erased. This type of display can have multiple SOURCE definitions.

4.6.4.2.7 VERTICAL_BAR

The actual format (visual elements) of the display is decided by the EDD application. This type dictates a vertical bar graph type of display. This type of display can have multiple bar charts. A bar chart is displayed for each variable.

4.6.4.3 EDDL application without full chart support

EDDL applications that do not support the graphical view of the chart should show the related variables in a standard numerical manner. For example, the data may be shown in tabular form. The format should be chosen by the EDD application developer.

4.6.4.4 Length and cycle time

The interval of time that is shown on the time axis is defined with the attribute LENGTH. The cycle time defines in ms the interval between the EDD application variable readouts. The EDD application updates the time axis with the time of the visible data. If the application cannot read as fast as defined by the cycle time it simply reads the data as fast as possible.

4.6.4.5 Data sources of a chart

A chart can display one or multiple curves. For each curve one variable is used. To support this, the CHART references one or multiple sources. The SOURCE can reference one or multiple variables.

Figure 42 shows a chart with one curve in a dialog. The curve is refreshed in a cycle time of 1 s.

```

MENU measuring_values
{
    LABEL "Measuring Values";
    STYLE DIALOG;
    ITEMS
    {
        primary_value_view
    }
}

MENU primary_value_view
{
    LABEL "Primary Measuring Value";
    STYLE PAGE;
    ITEMS
    {
        primary_value_chart
    }
}

VARIABLE primary_value
{
    LABEL "Primary Value";
    CLASS DYNAMIC;
    TYPE FLOAT;
    CONSTANT_UNIT "Bar";
}

SOURCE primary_value_source
{
    MEMBERS
    {
        PRIM_VAL, primary_value;
    }
}

CHART primary_value_chart
{
    LABEL "Primary Value";
    MEMBERS
    {
        CHART1, primary_value_source;
    }
}

```

Figure 42 – Example of a chart with one curve in a dialog

4.6.4.6 CHARTs with multiple SOURCES referencing the same AXIS

An EDD application shall combine SOURCES referencing the same AXIS. In case of CHARTs of TYPE HORIZONTAL_BAR and VERTICAL_BAR the different bars shall be displayed together with one axis. In case of a CHART of TYPE GAUGE, SCOPE, STRIP and SWEEP the curves shall be shown together in the chart area and only one axis shall be shown at a side of the chart area.

Figure 43 is an EDD example that shows the combining of two SOURCES. Figure 44 shows how an EDD application may display the chart.

```

VARIABLE primary_value
{
    LABEL "Primary Value";
    CLASS DYNAMIC;
    TYPE FLOAT;
    CONSTANT_UNIT "pH";
}

VARIABLE primary_value_undamped
{
    LABEL "Undamped PV";
    CLASS DYNAMIC;
    TYPE FLOAT;
    CONSTANT_UNIT "pH";
}

AXIS axis_0_14
{
    LABEL "axis 0-14";
    MIN_VALUE 0;
    MAX_VALUE 14;
}

SOURCE primary_value_stack1
{
    MEMBERS
    {
        PRIM_VAL_1, primary_value;
    }
    Y_AXIS axis_0_14;
    LINE_COLOR 0x0000FF; /*BLUE*/
}

SOURCE primary_value_stack2
{
    MEMBERS
    {
        PRIM_VAL_2, primary_value_undamped;
    }
    Y_AXIS axis_0_14;
    LINE_COLOR 0xFF0000; /*RED*/
}

CHART primary_value_stack_chart
{
    LABEL "Primary Value";
    MEMBERS
    {
        CHART1, primary_value_stack1, "ph1";
        CHART2, primary_value_stack2, "pH2";
    }
    TYPE VERTICAL_BAR;
}

```

Figure 43 – Example of a chart with two SOURCES

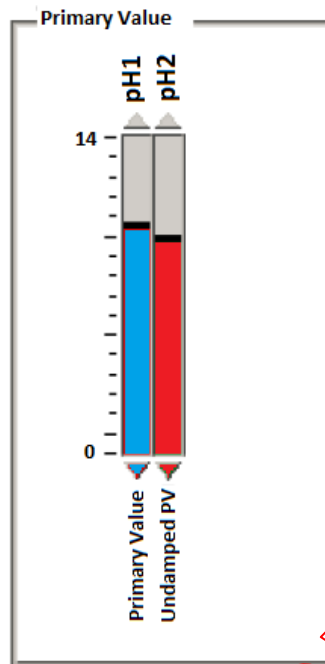


Figure 44 – Displaying example of a chart with two SOURCES

Figure 45 is an EDD example that shows the combining of three horizontal SOURCES. Figure 46 shows how an EDD application may display it.

```

VARIABLE primary_value
{
    LABEL "Primary Value";
    CLASS DYNAMIC;
    TYPE FLOAT;
    CONSTANT_UNIT "pH";
}

VARIABLE primary_value_undamped
{
    LABEL "Undamped PV";
    CLASS DYNAMIC;
    TYPE FLOAT;
    CONSTANT_UNIT "pH";
}

VARIABLE primary_value_non_temperature_compensated
{
    LABEL "PV non-TC";
    CLASS DYNAMIC;
    TYPE FLOAT;
    CONSTANT_UNIT "pH";
}

AXIS axis_0_14
{
    LABEL "axis 0-14";
    MIN_VALUE 0;
    MAX_VALUE 14;
}

AXIS axis_0_7
{
    LABEL "axis 0-7";
    MIN_VALUE 0;
    MAX_VALUE 7;
}

SOURCE primary_value_stack1
{
    MEMBERS
    {
        PRIM_VAL_1, primary_value;
    }
    Y_AXIS axis_0_14;
    LINE_COLOR 0x0000FF; /*BLUE*/
}

SOURCE primary_value_stack2
{
    MEMBERS
    {
        PRIM_VAL_2, primary_value_undamped;
    }
    Y_AXIS axis_0_14;
    LINE_COLOR 0xFF0000; /*RED*/
}

SOURCE primary_value_single
{
    MEMBERS
    {
        PRIM_VAL_3, primary_value_non_temperature_compensated;
    }
    Y_AXIS axis_0_7;
    LINE_COLOR 0x008000; /*GREEN*/
}

CHART primary_value_stack_chart
{
    LABEL "Primary Value";
    MEMBERS
    {
        CHART1, primary_value_stack1, "pH1";
        CHART2, primary_value_single, "pH non-TC";
        CHART3, primary_value_stack2, "pH2";
    }
    TYPE HORIZONTAL_BAR;
}

```

Figure 45 – Example of a chart with three horizontal bars

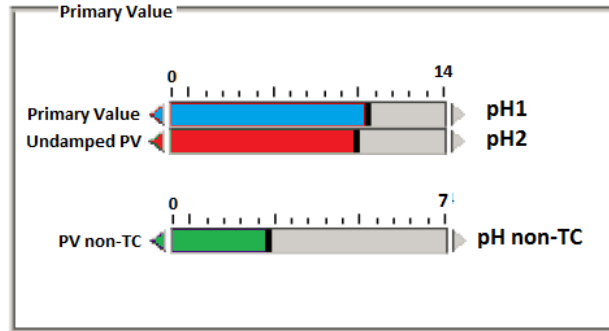


Figure 46 – Displaying example of a chart with three horizontal bars

4.6.4.7 Legend and help for the curves

The EDD can provide label and help information for CHARTs and its curves and axis. To build legends and help information for the curves the EDD application shall support the following rules.

The EDD application shall display the description if it is defined by the CHART members as a legend. The EDD application shall use the LABEL of the referenced SOURCE, if the description is not defined. An empty string constitutes a defined string.

In case SOURCES have multiple variables, the EDD application shall display the description if it is defined by SOURCE members as a legend. The EDD application shall use the LABEL of the referenced VARIABLE, if the description is not defined.

The EDD application shall provide the help information of CHART members to the user.

If no HELP attribute on the CHART members exists the help information of the SOURCES shall be used. If no HELP attribute on the SOURCES exists the help information of SOURCE members shall be used. If no HELP attribute exists for CHART, SOURCES and SOURCE members the help information of the VARIABLES shall be used.

4.6.4.8 Zooming and scrolling

The EDD application can support zooming and scrolling. Therefore, the EDD requires no support. The EDD application reads the variables of a chart and stores them in an own storage. When zooming and scrolling the EDD application just shows less, more or other points of the store data in the display area.

4.6.4.9 Actions

Optional INIT_ACTIONS and/or REFRESH_ACTIONS can be defined in a SOURCE. The variables that are referenced in the SOURCE should be set in the methods. The value can be calculated by using device variables and/or local variables.

If INIT_ACTIONS are defined, the EDD application calls these methods instead of reading the variables.

If REFRESH_ACTIONS are defined, the EDD application calls these methods cyclically in the CYCLE_TIME interval instead of reading the variables.

The same method can be referenced within the INIT_ACTIONS and REFRESH_ACTIONS method lists.

Figure 47 shows an example of how to use the methods in a chart. This example shows a chart in a dialog.

```

MENU measuring_values
{
    LABEL "Measuring Values";
    STYLE DIALOG;
    ITEMS
    {
        primary_value_view
    }
}

MENU primary_value_view
{
    LABEL "Primary Measuring Value";
    STYLE PAGE;
    ITEMS
    {
        primary_value_chart
    }
}

VARIABLE primary_value
{
    LABEL "Primary Value";
    CLASS DYNAMIC;
    TYPE FLOAT;
}

SOURCE primary_value_source
{
    LABEL "Primary";                                     // this label is used in the legend
    LINE_TYPE DATA1;
    EMPHASIS TRUE;
    Y_AXIS measuring_values_axis;
    MEMBERS
    {
        PRIM_VAL, primary_value;
    }
}

METHOD CalculationMethod
{
    LABEL "";
    DEFINITION
    {
        calculated_value = primary_value * 0.12345;
    }
}

AXIS measuring_values_axis
{
    LABEL "measurement value";
    MIN_VALUE 0;
    MAX_VALUE 100;
}

VARIABLE primary_value_unit
{
    LABEL "Primary Value Unit";
    CLASS CONTAINED;
    TYPE ENUMERATED(1)
    {
        { 32, [degC], [degC_help] },
        { 33, [degF], [degF_help] },
        { 35, [Kelvin], [Kelvin_help] }
    }
}

UNIT
{
    primary_value_unit:
    primary_value,
    calculated_value,
    measuring_values_axis
}

SOURCE calculated_value_source
{
    LABEL "calculated";                                   // this label is used in the legend
    LINE_TYPE DATA2;
}

```

```

Y_AXIS measuring_values_axis;
MEMBERS
{
    CALC_VAL, calculated_value;
}
INIT_ACTIONS {CalculationMethod}
REFRESH_ACTIONS {CalculationMethod}
}

CHART primary_value_chart
{
    LABEL "Primary Value";
    LENGTH 600000;
    TYPE SCOPE;
    WIDTH LARGE;
    HEIGHT SMALL;
    MEMBERS
    {
        GRAPH1, primary_value_source;
        GRAPH2, calculated_value_source;
    }
}

```

Figure 47 – Example of a chart in a dialog

4.6.5 GRAPH

4.6.5.1 Overview

A scalable GRAPH solution is supported by the EDDL applications using waveforms and axis definitions. Multiple waveforms can be defined that are based on one or multiple y-axes and one x-axis. Methods may be used for data retrieval and for scrolling and zooming.

The background colour of a GRAPH display is defined by the EDD application. To provide a common look and feel, it is recommended that the background be white. An EDD application should support the simultaneous display of at least six curves.

A graph is made of two main elements, one is the WAVEFORM and the other is the AXIS. The WAVEFORM provides the source and type of data, the emphasis to be provided, the visual elements and any actions to be performed when initialized or refreshed. A single graph can have multiple WAVEFORMs. This could include data as well as the lines on a graph, which display limits and markers. There is no upper limit defined for the number of WAVEFORMs in a GRAPH. The WAVEFORMs also contain a reference to a Y_AXIS definition. When using more than one WAVEFORM, the same Y_AXIS should be referenced. If the WAVEFORMs use different axes, each can have a different scaling and unit.

Figure 48 captures a graph and the visual elements, which are provided by the constructs defined in EDDL. Each of these elements will be shown in more detail in the relevant clauses.

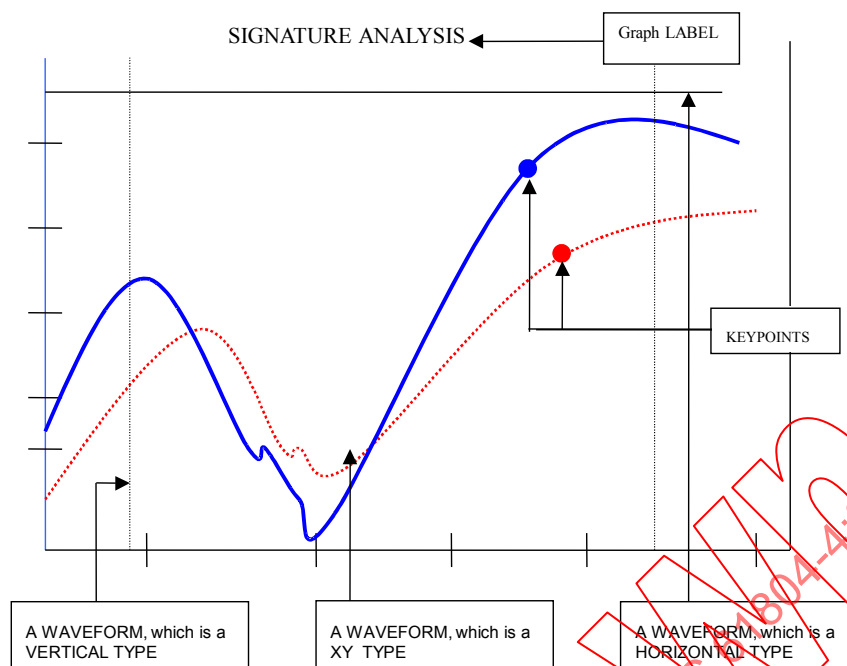


Figure 48 – A graph and the visual elements

4.6.5.2 Types

The GRAPH construct has WAVEFORM attributes, which define the data that can be shown on the GRAPH. The WAVEFORM can be used to plot limit values or envelopes in a display. The HORIZONTAL and VERTICAL types are used for this. The WAVEFORM construct provides the TYPE attributes that select between XY, YT, HORIZONTAL or VERTICAL.

The XY type provides a set of both X and Y point lists. The EDD developer should ensure that the position of the “x value” in the X list is the same as the position of the corresponding “y value” in the Y list. The number of points is also defined in the construct. If no number is defined, the EDD application sets this to the number of points in the list. This would be useful in instances where the number of points is not known.

A WAVEFORM with type YT provides a set of Y values. The initial X value and increments are defined in order to generate the subsequent X values. This would normally be used to represent waveforms, which are sampled on a periodic basis, so that X repeats at regular intervals. The number of points is defined in the waveform. In the absence of the number of points, the EDD application uses the number of Y values as the number of points.

A WAVEFORM with type HORIZONTAL allows the user to plot horizontal lines on the graphical display. This would normally be used to indicate maximum or minimum bounds, or a bounding envelope. The construct has a series of Y values, with each Y value specifying a horizontal line.

A WAVEFORM with type VERTICAL allows the user to plot vertical lines on the graphical display. This would normally be used to indicate maximum or minimum bounds, or a bounding envelope. The construct has a series of X values, with each X value specifying a vertical line.

The visual characteristics of a WAVEFORM are specified by the LINE_TYPE attribute. A WAVEFORM is visualized as a series of points when the LINE_TYPE attribute is specified as TRANSPARENT.

4.6.5.3 HANDLING

This attribute defines the type of operations that can be performed on the WAVEFORM (read, read/write or write).

4.6.5.4 KEY_POINTS

This attribute defines certain points emphasized on the graph. The X and Y values of the key point are provided in the WAVEFORM. The EDD application needs to provide emphasis on this point on the display. The determination of how the emphasis is provided is left to the EDD application.

4.6.5.5 Actions

4.6.5.5.1 INIT_ACTIONS

This attribute defines the set of actions that are to be executed before the initial display of the WAVEFORM. This is specified as references to METHOD instances, which need to be called in a particular order. If a METHOD fails or aborts for any reason then the display of the WAVEFORM is aborted. This construct may be used for the initial reading of the WAVEFORM from the device or a FILE and also any preprocessing needed (for example, averaging, or filtering of data), which is performed before the display.

4.6.5.5.2 REFRESH_ACTIONS

The REFRESH_ACTIONS attribute provides references to methods that are to be executed when changes are made to the X or Y axis or if a CYCLE_TIME is defined within the GRAPH each time the CYCLE_TIME elapses. The methods are executed in the order of the definition. If a method execution fails or aborts, the remaining methods are not executed and the GRAPH reverts back to the previous display.

The method may read the data in sections if the data size is so large that it cannot be transferred within one communication transfer.

Within the method, the visible area may be calculated and set on the axis with VIEW_MIN and VIEW_MAX. All the waveforms associated with the same axis shall be shown with the same area in the graph.

A simple example of a graph is shown in Figure 49.

4.6.5.5.3 EXIT_ACTIONS

The EXIT_ACTIONS are called if the waveform disappears from the screen.

Exit actions allow edited waveform data to be captured and manipulated as needed by the EDD.

Also if the device needs to be in any special (for example, diagnostic) mode to supply waveform data that mode may be reset upon closure of the graph.

Figure 49 shows an example of a graph.

```

VARIABLE      x_value
{
    LABEL "...";
    HELP "...";
    CLASS DYNAMIC;
    TYPE FLOAT;
    CONSTANT_UNIT [degC];
}

ARRAY          x_data
{
    LABEL "x-data";
    NUMBER_OF_ELEMENTS 1000;
    TYPE x_value;
}

VARIABLE      y_value
{
    LABEL "...";
    HELP "...";
    CLASS DYNAMIC;
    TYPE FLOAT;
    CONSTANT_UNIT [degC];
}

ARRAY          y_data
{
    LABEL "y-data";
    NUMBER_OF_ELEMENTS 1000;
    TYPE y_value;
}

VARIABLE      x_min_value
{
    LABEL "...";
    HELP "...";
    CLASS LOCAL;
    TYPE FLOAT;
}

AXIS x_axis_signature
{
    LABEL "x axis";
    MIN_VALUE DEFAULT X_MIN_VALUE;
    MAX_VALUE DEFAULT X_MAX_VALUE;
}

AXIS y_axis_signature
{
    LABEL "y axis";
    MIN_VALUE DEFAULT Y_MIN_VALUE;
    MAX_VALUE DEFAULT Y_MAX_VALUE;
}

WAVEFORM value_signature1
{
    TYPE XY
    {
        X_VALUES      { x_data }
        Y_VALUES      { y_data }
    }
    INIT_ACTIONS { read_first_signature }
    REFRESH_ACTIONS { read_signature }
    Y_AXIS y_axis_signature;
}

x_max_value      LIKE x_min_value;
device_x_min_value LIKE x_min_value;
device_x_max_value LIKE x_min_value;

y_min_value      LIKE x_min_value;
y_max_value      LIKE x_min_value;
device_y_min_value LIKE x_min_value;
device_y_max_value LIKE x_min_value;

METHOD read_first_signature
{
    DEFINITION
    {

```

```

        // read data from the device depending predefined default
        // MIN_VALUE and MAX_VALUE of the x and y axis
        x_min_value = DEFAULT_X_MIN_VALUE;
        x_max_value = DEFAULT_X_MAX_VALUE;
        y_min_value = DEFAULT_Y_MIN_VALUE;
        y_max_value = DEFAULT_Y_MAX_VALUE;
        WriteCommand (write_data_area);
        ReadCommand (read_data);
        x_axis_signature.MIN_VALUE = device_x_min_value;
        x_axis_signature.MAX_VALUE = device_x_max_value;
        y_axis_signature.MIN_VALUE = device_y_min_value;
        y_axis_signature.MAX_VALUE = device_y_max_value;
    }
}

METHOD read_signature
{
    DEFINITION
    {
        // read data from the device depending of the current MIN_VALUE and MAX_VALUE
        // of the x axis
        x_min_value = x_axis_signature.VIEW_MIN;
        x_max_value = x_axis_signature.VIEW_MAX;
        y_min_value = y_axis_signature.VIEW_MIN;
        y_max_value = y_axis_signature.VIEW_MAX;
        WriteCommand (write_data_area);
        ReadCommand (read_data);
        if (device_x_min_value < x_min_value)
            x_axis_signature.VIEW_MIN = device_x_min_value;
        if (device_x_max_value > x_max_value)
            x_axis_signature.VIEW_MAX = device_x_max_value;
        if (device_y_min_value < y_min_value)
            y_axis_signature.VIEW_MIN = device_y_min_value;
        if (device_y_max_value > y_max_value)
            y_axis_signature.VIEW_MAX = device_y_max_value;
    }
}

GRAPH valve_signature
{
    MEMBERS
    {
        VAL SIG, valve_signature;
    }
    X_AXIS x_axis_signature;
}

```

Figure 49 – Example of a graph

4.6.5.6 Axis

The Y_AXIS definition is provided by the WAVEFORM construct whereas the X_AXIS definition is provided by the GRAPH construct itself.

This attribute provides a reference to the AXIS. When displayed, this AXIS is to be drawn with the waveform. If this is not defined, the EDD application constructs the AXIS based on maximum and minimum values and any other internal rules it may have.

The AXIS construct defines how to construct the X or Y axis, which shall be displayed on a GRAPH or CHART. The AXIS has attributes which define its maximum and minimum values. In addition, there is a “SCALING” attribute, which defines whether the axis should be scaled “LINEAR” or “LOGARITHMIC”. Logarithmic scaling is in base 10.

The EDD developer can define a LABEL that can be displayed with the AXIS and also a string, which specifies the units in which the values are displayed on the GRAPH.

Figure 50 shows an EDD example of a common axis for WAVEFORM w1 and w2 and a second axis for WAVEFORM w3.

```

AXIS    x1 { }
AXIS    y1 { }
AXIS    y2 { }

GRAPH graph1
{
    MEMBERS
    {
        w1, w1;
        w2, w2;
        w3, w3;
    }
    X_AXIS x1;
}

WAVEFORM w1
{
    Y_AXIS y1;
    TYPE XY
    {
        Y_VALES {w1_values}
        X_INCREMENT 1
        X_INITIAL 0
    }
}

WAVEFORM w2
{
    Y_AXIS y1;
    TYPE XY
    {
        Y_VALES {w2_values}
        X_INCREMENT 1
        X_INITIAL 0
    }
}

WAVEFORM w3
{
    Y_AXIS y2;
    TYPE XY
    {
        Y_VALES {w3_values}
        X_INCREMENT 1
        X_INITIAL 0
    }
}

```

Figure 50 – Multiple used axes**4.6.5.7 EDDL application without graphical support**

If an EDDL application does not support the graphical view of a graph, then the values of the related variables can be shown in a table.

4.6.5.8 Standard usage

A GRAPH is invoked by a MENU or METHOD. A GRAPH references one or more WAVEFORMs, each representing a unique curve on the GRAPH. Each WAVEFORM describes the source of the data points. The EDD application executes the INIT_ACTIONS prior to displaying the WAVEFORM on the GRAPH. This permits calculations of the data prior to rendering. REFRESH_ACTIONS are executed by the EDD application in order to specify the visible area of the GRAPH, permitting a zoomed visualization.

4.6.5.9 Integration

A GRAPH is rendered by an EDD application through a MENU or METHOD.

A MENU may contain one or more GRAPHS. The graph can be integrated into all visible menus. It is possible to use one or more graphs together with any input and output fields within one menu. The EDD application executes the INIT_ACTIONS prior to rendering the GRAPH.

A GRAPH is rendered by a METHOD via Builtins. The Builtin MenuDisplay() is used to render the GRAPH. The INIT_ACTIONS of the GRAPH are called prior to displaying the menu.

4.6.5.10 GRAPH with multiple WAVEFORMs referencing same Y_AXIS

An EDD application shall combine WAVEFORMs referencing same Y_AXIS. The curves shall be shown together in the graph area and only one y-axis shall be shown at a side of the graph area.

4.6.5.11 Multiple waveforms and legends

The EDD application should display a legend in order to differentiate between different multiple waveforms on a single graph. The label for each waveform is derived from the LABEL attribute of the WAVEFORM. WAVEFORMs without a LABEL attribute should not be displayed on the legend.

4.6.5.12 Editable graphs

4.6.5.12.1 General

The HANDLING attribute of a WAVEFORM determines whether the user can edit the waveform. The EDD application should provide a mechanism for the user to change the waveform (direct drag and click, table entry, etc.) and a mechanism for the user to indicate that the waveform modification is complete. For GRAPHS that are generated by a MENU, the EDD application should provide a mechanism for the user to begin editing the WAVEFORM. The EDD application should update the WAVEFORMs data when the user indicates that the waveform modification is complete (for example, the save button). The EDD application should provide a mechanism for the user to restore the original WAVEFORM without updating the data (for example, the cancel button).

For a GRAPH with a CYCLE_TIME definition, the EDD application shall suspend the source value read from the device when the GRAPH is being edited.

4.6.5.12.2 POST_EDIT_ACTIONS on variables

POST_EDIT_ACTIONS should be called for each change of a graph waveform. Data inputs can be checked and any data can be modified depending on a change.

4.6.5.13 Legend and help of the curves

The EDD can provide label and help information for GRAPHS and its curves and axis. To build legends and help information for the curves the EDD application shall support following rules.

The EDD application shall display the description if it is defined in the GRAPH members as a legend. The EDD application shall use the LABEL of the referenced WAVEFORM, if the description is not defined. An empty string constitutes a defined string.

The EDD application shall provide help information of GRAPH members to the user.

If no HELP attribute on GRAPH members exists the help information of the WAVEFORMs shall be used.

4.6.5.14 Device-supported zooming and scrolling

The EDD application should support zooming and scrolling without support in the EDD. The EDD application shows fewer or more points, or data from the currently not displayed part of the WAVEFORM on the display area. In addition to that, the REFRESH_ACTIONS in the EDD can support scrolling and zooming. The EDD application calls the REFRESH_ACTIONS if the user scrolls into an area that is not stored in the waveform data. It also calls the REFRESH_ACTIONS if the user is zooming and more points should be shown. Information about the position and zooming can be read from the axis with the VIEW_MIN and VIEW_MAX attributes.

Figure 51 shows an EDD example with device supported zooming and scrolling.

```

VARIABLE      y_value
{
    LABEL "...";
    HELP "...";
    CLASS DYNAMIC;
    TYPE FLOAT;
    CONSTANT_UNIT [degC];
}

ARRAY          y_data
{
    NUMBER_OF_ELEMENTS 1000;
    TYPE y_value;
}

COMMAND read_data
{
    SLOT ...; INDEX...;
    OPERATION READ;
    TRANSACTION
    {
        REPLY
        {
            device_t_min_value, device_t_max_value,
            device_y_min_value, device_y_max_value,
            y_data
        }
    }
}

COMMAND read_current_data
{
    SLOT ...; INDEX...;
    OPERATION READ;
    TRANSACTION
    {
        REPLY
        {
            device_t_min_value, device_t_max_value,
            device_y_min_value, device_y_max_value,
            y_data
        }
    }
}

COMMAND write_data_area
{
    SLOT ...; INDEX...;
    OPERATION WRITE;
    TRANSACTION
    {
        REQUEST
        {
            t_min_value, t_max_value,
            y_min_value, y_max_value
        }
    }
}

VARIABLE y_increment
{
    LABEL "..."; HELP "...";
}

```

```

        CLASS LOCAL;
        TYPE TIME;
    }

VARIABLE current_date_time1
{
    LABEL "..."; HELP "...";
    CLASS LOCAL;
    TYPE DATE_AND_TIME;
}

VARIABLE t_min_value
{
    LABEL "..."; HELP "...";
    CLASS LOCAL;
    TYPE DATE_AND_TIME;
}

t_max_value LIKE t_min_value;
t_device_min_value LIKE t_min_value;
t_device_max_value LIKE t_min_value;

VARIABLE y_min_value
{
    LABEL "..."; HELP "...";
    CLASS LOCAL;
    TYPE FLOAT;
}

y_max_value LIKE min_value;
device_y_min_value LIKE min_value;
device_y_max_value LIKE min_value;

AXIS y_axis_signature
{
    LABEL          "...";
    HELP           "...";
    MIN_VALUE      y_min_value;
    MAX_VALUE      y_max_value;
    SCALING        LINEAR;
}

AXIS t_axis_signature
{
    LABEL          "...";
    HELP           "...";
    MIN_VALUE      t_min_value;
    MAX_VALUE      t_max_value;
    SCALING        LINEAR;
}

WAVEFORM value_signature1
{
    LABEL          "...";
    HELP           "...";
    HANDLING       READ;    // alternative READ & WRITE
    LINE_TYPE      DATA1;
    EMPHASIS       TRUE;

    TYPE           XY
    {
        X_INITIAL      current_date_time1;    // starting time in milliseconds
        X_INCREMENT     y_increment;          // time between two points in ms
        Y_VALUES        { y_data }
    }

    Y_AXIS          y_axis_signature;

    INIT_ACTIONS { init_signature}
    REFRESH_ACTIONS { refresh_signature}
}

METHOD init_signature
{
    DEFINITION
    {
        // read current data and stored area t_device_min_value,
        // t_device_max_value, y_device_min_value, y_device_max_value
        read_command (read_current_data);
    }
}

```

```

        // calculation of the waveform data
        value_signature1.X_INITIAL = t_device_max_value;
        value_signature1.X_INCREMENT = (t_device_max_value-t_device_min_value)/
            ( y_data.NUMBER_OF_ELEMENTS - 1 );

        // set the visible area to the area from the device
        t_axis_signature.MIN_VALUE = t_device_min_value;
        t_axis_signature.MAX_VALUE = t_device_max_value;
        y_axis_signature.MIN_VALUE = y_device_min_value;
        y_axis_signature.MAX_VALUE = y_device_max_value;
    }
}

METHOD refresh_signature
{
    DEFINITION
    {
        // read data from the device depending of the current
        // MIN_VALUE and MAX_VALUE of the time axis
        t_min_value = t_axis_signature.MIN_VALUE;

        t_max_value = t_axis_signature.MAX_VALUE;
        y_min_value = y_axis_signature.MIN_VALUE;
        y_max_value = y_axis_signature.MAX_VALUE;

        // writing requested t_min_value, t_max_value, y_min_value, y_max_value
        write_command (write_data_area);

        // reading points and area      t_device_min_value, t_device_max_value,
        //                               y_device_min_value, y_device_max_value from the
device
        read_command (read_data);

        // reduce the visible area to the from the device supported area
        if (t_device_min_value > t_min_value)
            t_axis_signature.MIN_VALUE = t_device_min_value;
        if (t_device_max_value < t_max_value)
            t_axis_signature.MAX_VALUE = t_device_max_value;
        if (y_device_min_value > y_min_value)
            y_axis_signature.MIN_VALUE = y_device_min_value;
        if (y_device_max_value < y_max_value)
            y_axis_signature.MAX_VALUE = y_device_max_value;
    }
}

```

Figure 51 – EDD with device-supported zooming and scrolling

4.6.6 AXIS

Axes are used for charts and graphs. In a chart with curves, the x-axis is controlled by the EDD application. In a graph, only one x-axis can be referenced.

The y-axes of a chart are referenced in the SOURCE. The y-axes of a graph are referenced in the WAVEFORM. If some sources or waveforms reference to the same y-axis within one chart or graph, the EDD application should draw the y-axis only once.

The MIN_VALUE and MAX_VALUE are optional attributes. If they are not defined in the axis, the EDD application determines them such that MIN_VALUE and MAX_VALUE shall be continuously calculated with e.g. 150 % of the data being displayed (i.e. auto-scaled). In the case of a graph, the EDD application can build the minimum and maximum X and Y value of the arrays. In the case of a chart, the EDD application can read some values and build an initial minimum and maximum. The EDD application should recalculate the axes if the value goes out of range.

VIEW_MIN and VIEW_MAX are attributes that contain the current viewing area. The EDD application sets them before the INIT_ACTIONS are called and after the user has changed the zooming or positioning. Within a method, the VIEW_MIN and VIEW_MAX can be read and altered, for example, if the user sets the position to an area outside the available data, the method can set it to existing values.

The LABEL is used to give the axis a legend. If a CONSTANT_UNIT is defined, this unit will be used. However, if the variable of the axis is related in a UNIT relation, this unit is used. Otherwise, the axis has no unit.

The SCALING of an axis can be LINEAR or LOGARITHMIC. The SCALING default is LINEAR.

The EDD application displays absolute or relative time stamps on the x-axis of a chart.

4.6.7 IMAGE

The IMAGE basic construct is used to display images in windows, dialogs, pages and groups. If an EDD application cannot display the image because of the required display space, the EDD application can show the LABEL instead of the image. The EDD application shall display IMAGES having transparency with transparency against the background of its container.

For handhelds the PATH attribute can additionally have country code zz, see 4.2.

Figure 52 shows an image definition.

```
IMAGE Sensor_Diagram
{
    LABEL "Sensor Diagram";
    HELP "This Diagram shows the sensor characteristic";
    PATH "SenDiaEn.jpg\
        |zz|SenDiaEn_LowRes.jpg\
        |de|SenDiaDe.jpg\
        |de zz|SenDiaDe_LowRes.jpg";
}
```

Figure 52 – EDD example of an IMAGE

An IMAGE definition can be used to invoke a METHOD or a MENU of STYLE DIALOG or WINDOW with the LINK attribute

Figure 53 shows an EDD example of using the LINK attribute in an image.

```
MENU
{
    ITEMS
    {
        ...,
        Self_Test_Image (INLINE)
        ...
    }
}

IMAGE Self_Test_Image
{
    LABEL "Self Test";
    HELP "This Method execute the self test function of the device which needs some time";
    PATH "SelfTestImage.jpg";
    LINK Self_Test_Menu;
}

MENU Self_Test_Menu
{
    ...
}
```

Figure 53 – EDD example of an IMAGE with the LINK attribute

Image formats in Table 7 are allowed to be used within an image.

Table 7 – Image formats

Format	Description
JPG, JPEG	Format according to ISO/IEC 10918
PNG	Portable Network Graphics format according to ISO/IEC 15948
GIF	Graphics Interchange Format

Images for multiple locales may be specified using the same localization technique used when specifying string literals.

EXAMPLE: PATH "|en|english.gif|de|german.gif"

This example specifies an image file to be used when the application is employing English and another when German is employed.

4.6.8 GRID

GRID describes a matrix of data from a device to be displayed by the EDD application. A GRID is used to display vectors of data along with the heading or description of the data in that vector. The vectors are displayed horizontally (rows) or vertically (columns) as specified by the ORIENTATION attribute.

Figure 54 shows a GRID example.

```

VARIABLE PeakType
{
    LABEL "Peak Type";
    CLASS DEVICE;
    TYPE ENUMERATED
    {
        { 1, "False Echo"},
        { 2, "Button Echo" },
        { 3, "Unkown" }
    }
}

ARRAY arrPeakType
{
    NUMBER_OF_ELEMENTS 10;
    TYPE PeakType;
}

VARIABLE PeakDistance
{
    LABEL "Peak Distance";
    CLASS DEVICE;
    TYPE FLOAT
    {
        DEFAULT_VALUE 1.0;
    }
}

ARRAY arrPeakDistance
{
    LABEL "Peak Distance";
    NUMBER_OF_ELEMENTS 10;
    TYPE PeakDistance;
}

VARIABLE PeakAmplitude
{
    LABEL "Device Amplitude";
    CLASS DEVICE;
    TYPE FLOAT
    {
        DEFAULT_VALUE 1.0;
    }
}

```

```

ARRAY arrPeakAmplitude
{
    LABEL "Peak Amplitudes";
    NUMBER_OF_ELEMENTS 10;
    TYPE PeakAmplitude;
}

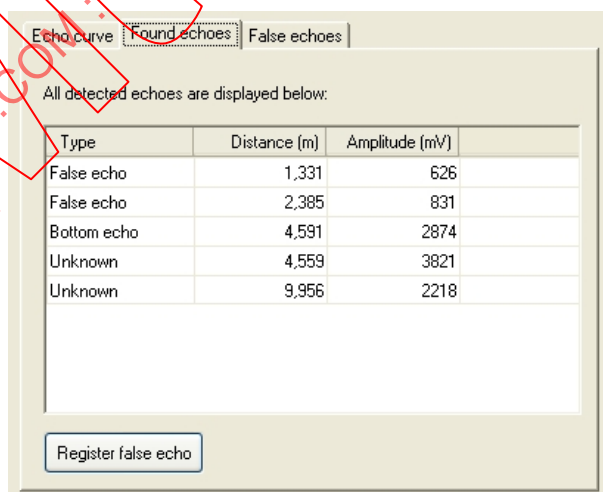
MENU DeviceEcho
{
    LABEL "Device Echo";
    ITEMS
    {
        EchoCurve,
        FoundEcho,
        FalseEchos
    }
}

MENU FoundEcho
{
    LABEL "Found echoes";
    STYLE PAGE;
    ITEMS
    {
        GridFoundEcho,
        RegisterFalseEchos
    }
}

GRID GridFoundEchos
{
    LABEL "All detected echoes are displayed below";
    VALIDITY IF (varModelCode == 4711) { TRUE; } ELSE { FALSE; }
    VECTORS
    {
        {"Type", arrPeakType},
        {"Distance (m)", arrPeakDistance},
        {"Amplitude (mV)", arrPeakAmplitude}
    }
}
    
```

Figure 54 – EDD example of a GRID

Figure 55 shows a hard copy of the result of the EDD example above.



The screenshot shows a software interface with three tabs: "Echo curve", "Found echoes", and "False echoes". The "Found echoes" tab is selected. Below the tabs, a label reads "All detected echoes are displayed below:". Underneath this label is a table with three columns: "Type", "Distance (m)", and "Amplitude (mV)". The table contains five rows of data. At the bottom of the interface, there is a button labeled "Register false echo".

Type	Distance (m)	Amplitude (mV)
False echo	1,331	626
False echo	2,385	831
Bottom echo	4,591	2874
Unknown	4,559	3821
Unknown	9,956	2218

Figure 55 – Result of the EDD example

5 EDDL data description

5.1 Variables

5.1.1 VARIABLE TYPEs

5.1.1.1 BIT_ENUMERATED

Multiple bits can be packaged in a single-bit enumerated variable. Each bit could have a distinct meaning. If enforcement is required, then ENUMERATED should be used instead of BIT_ENUMERATED.

Figure 56 shows an EDD example of a wrong usage of a BIT_ENUMERATED variable.

```
VARIABLE Measuring_Mode
{
    LABEL " Measuring Mode";
    TYPE BIT_ENUMERATED
    {
        { 0x08, "Massflow"},
        { 0x11, "Flow" }      /* this is not referencing a single bit!!! */
    }                       /* This is not allowed */
}
```

Figure 56 – Wrong usage of a BIT_ENUMERATED variable

For this example the type ENUMERATED is recommended because both alternatives are not allowed at the same time.

Figure 57 shows an EDD example with a VARIABLE of type ENUMERATED.

```
VARIABLE Measuring_Mode
{
    LABEL " Measuring Mode";
    TYPE ENUMERATED
    {
        { 8, "Massflow"},
        { 17, "Flow" }      /* this is correct */
    }
}
```

Figure 57 – Usage of ENUMERATED instead of BIT_ENUMERATED

5.1.1.2 ASCII, EUC, PACKED_ASCII, OCTET, PASSWORD, VISIBLE

The EDD application shall display these data types as strings. Depending of the data type, the trailing characters shall be treated as shown in Table 8.

Table 8 – String handling

TYPE	Termination	Fill character	Trimming trailing spaces
ASCII	Null	Null	No
EUC	Null	Null	No
PACKED_ASCII	Not terminated	Space	No
OCTET	Not terminated	Space	No
PASSWORD	HART: Null, FOUNDATION fieldbus: not terminated	HART: Null, FOUNDATION fieldbus: space	No
VISIBLE	Not terminated	Space	Yes

5.1.2 VARIABLE CLASS

5.1.2.1 Overview

The CLASS attribute specifies the usage of the variable. Some CLASSES have specific requirements for the EDD application.

5.1.2.2 CLASS OPTIONAL

A device profile defines variables as mandatory or optional. If a device supports a feature it shall support it in the way defined by the optional variable.

If a device does not support an optional variable the EDD application shall not display the value of this variable. The EDD application may hide the whole variable in that case.

If a method reads a not supported optional variable the VARIABLE_STATUS is set to VARIABLE_STATUS_NOT_SUPPORTED and the value of the variable does not exist. Before the method accesses the variable value the method shall check the VARIABLE_STATUS otherwise the method aborts.

If a method writes a not supported optional variable the VARIABLE_STATUS is set to VARIABLE_STATUS_NOT_SUPPORTED.

5.1.3 VARIABLE ACTIONS

5.1.3.1 PRE_EDIT_ACTIONS

PRE_EDIT_ACTIONS should be used to display information or warnings to the user. They should not be used for any manipulations of variables. The methods are called only if the user wants to edit the variable.

If a METHOD of PRE_EDIT_ACTIONS aborts the EDD application should give the user an indication of the abnormal process and the edit shall be cancelled.

5.1.3.2 POST_EDIT_ACTIONS

POST_EDIT_ACTIONS should be used to display information or warnings or to manipulate variables. The methods are called after the value of a variable is changed.

If a METHOD of POST_EDIT_ACTIONS aborts the EDD application should give the user an indication of the abnormal process.

5.1.3.3 REFRESH_ACTIONS

REFRESH_ACTIONS shall be used to calculate the value of the variable using other variables. REFRESH_ACTIONS shall not be used to modify other variables or to display anything. The methods shall be called every time before the variable needs to be refreshed in conditional expressions, for display purpose, etc.

5.2 EDDL application stored device data

5.2.1 Overview

A number of complex device types need access to historical performance-related data to assess the current operational condition. Two examples illustrate these kinds of devices: valve-positioners and radar level gauges.

In the case of a valve-positioner, a baseline signature is used when installation is complete. That signature is compared to the current signature at a later date. Significant differences in the signatures may indicate that maintenance is required.

In the case of a radar gauge, a number of return signals may be recorded under differing operating conditions (for example, different tank levels or different equipment in operation). Examination and/or comparison of these signals may allow gauge operation to be optimized or detect an unexpected change in operating conditions.

The EDD developer can specify data that is to be stored by the EDDL application. This data can be recalled at a later date for assessing the performance and/or condition of the device.

5.2.2 FILE

The FILE construct allows an EDD developer to specify data that is to be stored for later use. Like a COLLECTION it makes no difference between using the data directly and using over the file reference. In both cases, the same data is accessed. The difference between FILE and COLLECTION is that the data referenced by FILES is stored.

Persistent data is associated with one, and only one, device instance. Consider a system with four identical valve-positioners. The same EDD is used for all four devices. However, there are four different sets of instance data, one for each valve-positioner. When the FILE construct is employed, there is persistent data stored by the EDD application, which is associated with each device as well. If the FILE construct is used to store the device's valve signature then that signature is available at a later date. The signature for valve-positioner #2 is not available when the EDD is being used with valve-positioner #1.

The FILE construct only specifies the structure of the data to be stored. It does not specify how the EDD application stores the data or when the data is loaded into the local memory. The EDD application can load the data of the FILE when instantiating the device or can provide the data when demanded by the EDD. The persistent data may be stored in a flash memory or on a hard disc, on the computer executing the EDD application or on a file server. The FILE may be stored as a flat file in the operating systems file structure or it may be embedded into the database of the EDD application. In any case, access to all the data of the FILE is available once the EDD is instantiated. No low level open, close, read or write file commands need be called by the EDD.

The EDD developer defines the FILE's data schema or structure. The FILE has a list of members that can be any combination of data items, e.g. VARIABLES, ARRAYS, RECORDS, COLLECTIONS, or LISTS. This flexibility allows the EDD developer to store a fixed set of data by using only VARIABLES, ARRAYS, RECORDS, and COLLECTIONS. The EDD developer can also store a varying amount of data using the LIST construct (along with the VARIABLE, ARRAY, RECORD and COLLECTION constructs).

Figure 58 is a simple example of a FILE declaration. In this case, the EDDL application is requested to support a file named "reference_valve_signature".

```

VARIABLE valve_position
{
  TYPE FLOAT;
}

ARRAY valve_stroke
{
  TYPE valve_position;
  NUMBER_OF_ELEMENTS 1000;
}

VARIABLE valve_signature_comment
{
  TYPE ASCII(64);
}

FILE reference_valve_signature
{
  MEMBERS
  {
    COMMENT, valve_signature_comment;
    STROKE, valve_stroke;
  }
}

```

Figure 58 – Example of a file declaration

This FILE contains a brief note ("valve_signature_comment") about the signature and the signature itself ("valve_stroke"). The signature is an array of 1 000 equally spaced valve position samples.

This is a simple example. In a real application, the position samples may not be equally spaced, and thus a sample time will be needed. If that is the case, another array of sample times will be needed. Additional information such as date, time of day, operator, etc. can be added.

Members of a FILE are accessed via the dot notation in the same way a COLLECTION member is referenced. Thus

```
reference_valve_signature.STROKE[10];
```

would access the 10th value in the signature array or valve_stroke[10]. It is also easy to plot this signature and compare it with the current signature. While this would normally be done by a method, it is also possible to display it via a MENU.

Figure 59 shows a GRAPH in a MENU.

```

ARRAY current_stroke { TYPE valve_position; NUMBER_OF_ELEMENTS 1000; }

VARIABLE sample_interval { TYPE FLOAT; CONSTANT_UNITS "ms"; }

WAVEFORM stroke_now
{
    TYPE YT
    {
        X_INITIAL 0;
        X_INCREMENT sample_interval;
        Y_VALUES {current_stroke }
    }
}

WAVEFORM ref_stroke
{
    TYPE YT
    {
        X_INITIAL 0;
        X_INCREMENT sample_interval;
        Y_VALUES { valve_stroke }
    }
}

GRAPH compare_stroke
{
    MEMBERS
    {
        NOW, stroke_now;
        THEN, ref_stroke;
    }
}

MENU compare_strokes
{
    LABEL "CompStrokes";
    ITEMS
    {
        compare_stroke
    }
}

```

Figure 59 – Example of comparing valve signatures

In the example shown in Figure 59, data sampled during a recent stroke of the valve is stored in "current_stroke" and the sample interval (the inverse of the sample rate) indicates the time spacing between sample points. Two WAVEFORMs and a GRAPH are declared, along with a MENU containing the GRAPH. The GRAPH will be displayed by the EDDL application when the MENU is accessed.

When the graph is displayed, the EDD applications recognize that "valve_stroke" is a member of a FILE. Consequently, when the GRAPH is displayed data should be automatically provided from the "reference_valve_signature" FILE.

5.2.3 LIST

LIST is a variable length, insertable array. Each element stored in the list has exactly the same structure. That structure is specified using the mandatory TYPE attribute. The TYPE attribute can refer to any legal EDD data item or structure (for example, VARIABLE, ARRAY, RECORD, COLLECTION, LIST). Individual elements in the LIST are accessed using the square bracket notation, as when accessing ARRAY elements.

An example showing the use of a LIST in a FILE is given in Figure 60. In this example, the LIST (and thus the FILE) can grow in size as additional interesting radar waveforms, which document the operation of the level gauge, are stored.

```

VARIABLE gauge_location      { TYPE ASCII(32); }
VARIABLE tag                 { TYPE ASCII(32); }

VARIABLE date_taken          { TYPE DATE; }
VARIABLE operator            { TYPE ASCII(32); }
VARIABLE operating_conditions { TYPE ASCII(64); }
VARIABLE sample_interval     { TYPE FLOAT; CONSTANT_UNITS "ms"; }

VARIABLE sample              { TYPE UNSIGNED_INTEGER(2); }
ARRAY    echo_signal         { TYPE sample; NUMBER_OF_ELEMENTS 4096; }

COLLECTION signal_info
{
    MEMBERS
    {
        DATE_STAMP, date_taken;
        TAKEN_BY,   operator;
        NOTES,      operating_conditions;
        DT,         sample_interval;
        SIGNAL,     echo_signal;
    }
}

LIST recorded_signals { TYPE signal_info; }

FILE stored_signals
{
    MEMBERS
    {
        LOCATION,    gauge_location;
        INSTR_TAG,   tag;
        SIGNALS,     recorded_signals;
    }
}

```

Figure 60 – Example of more complex file declaration

In the example of Figure 60, basic information ("gauge_location", "tag") about the level gauge is stored along with a series of radar signals ("recorded_signals"). In this example, "signal_info" is used as a type definition for the list. The referencing "signal_info" in the EDD will not access the list. The list can only be accessed by using the square bracket notation. The following two references refer to the same data.

```

stored_signals.SIGNALS[10]
recorded_signals[10]

```

However, the following references do not access the same information.

```

stored_signals.SIGNALS[10].SIGNAL
echo_signal;

```

The "echo_signal" defines the structure for one part of a list element and, in both cases, the amount of data referred to is the same (in both cases an array of 4 096, 2-byte unsigned integers). However, "echo_signal" will be null unless data is placed into it (for example, by loading it with data from the field device).

Figure 61 shows an example of a method that is used for reviewing the stored radar signals. In this example, the LIST of stored waveforms is displayed sequentially. The "i" variable is used to step through the entire LIST, just like an iterator.

```

WAVEFORM one_echo
{
    TYPE YT
    {
        X_INITIAL 0;
        X_INCREMENT sample_interval;
        Y_VALUES { echo_signal }
    }
}

GRAPH review_radar_signal
{
    MEMBERS
    {
        PING, one_echo;
    }
}

MENU review_radar_signal_menu
{
    LABEL "Radar Signal";
    STYLE WINDOW;
    ITEMS
    {
        review_radar_signal
    }
}

/*
*****
* now for a method that reviews the stored radar signals (ping).
*
*/
METHOD reviewStoredPings
{
    LABEL "SignalHistory";
    DEFINITION
    {
        int i,
        ans;          /*The users answer to select from list*/
        long selection;
        long varid[5]; /*used in the acknowledge Builtin calls*/

        i = 0;

        varid[0] = VARID( date_taken );
        varid[1] = VARID( operator );
        varid[2] = VARID( operating_conditions );
        varid[3] = VARID( sample_interval );
        varid[4] = VARID( echo_signal );

        do
        {
            /* get the current record so we can display it */
            date_taken      = stored_signals.SIGNALS[i].DATE_STAMP;
            operator        = stored_signals.SIGNALS[i].TAKE_BY;
            operating_conditions = stored_signals.SIGNALS[i].NOTES;
            sample_interval = stored_signals.SIGNALS[i].DT;
            echo_signal     = stored_signals.SIGNALS[i].SIGNAL;

            /* display the graph and the annotations. */

            MenuDisplay (review_radar_signal_menu,"OK", selection);
            acknowledge ( "Radar Signal by %{1} \nDescription %{2}" varid);

            ans = select_from_list ("do you want to see the next radar signal",
                                   "Yes;No");

            if (ans != 0)
            {
                break;
            }
            i++;
        } while ( i < recorded_signals.COUNT);
    }
}

```

Figure 61 – Example of reviewing the stored radar signals

This example has a number of interesting features. Firstly, the Builtin MenuDisplay allows the use of menus of STYLE WINDOW or DIALOG to be used within methods.

Each iteration also calls "acknowledge()". Within a method, the graph object is separate and asynchronously displayed. This allows the EDD developer to use the acknowledge (delay and put_message) Builtins to interact with the user in exactly the same way as always used in METHODS.

The acknowledge call displays the operator and comments on the recorded signal. The date is not currently displayed. Displaying the date is possible with some work but the implementation is not shown in this example.

Lastly, it should be noticed that the "recorded_signals.COUNT" COUNT (along with FIRST, LAST, AND CAPACITY) is an attribute inherently available for all LISTS. COUNT can be used to detect when the end of the LIST is reached. Any access on non existing LIST elements in METHODS causes process abort.

If COUNT is specified in a LIST it defines the size that it has at creation time of the instance device data. If it is not defined, the list is empty.

A list can be filled within a method by calling ListInsert Builtin or by reading with a command that has a variable with attributes INDEX and INFO. Elements of a list can be deleted in a method by calling ListDeleteElementAt.

The index of a list starts with 0 and is always continuous. If an element of a list is deleted, the indexes of the following elements will be decremented. If an element is inserted the indexes of the following elements will be incremented.

COUNT can be used to get the current number of elements of a list. It shall be automatically updated with ListInsert or ListDeleteElementAt by the EDD application.

Figure 62 is more involved. It takes a measurement and reads the radar signal back for the operator to review. Command 128 starts a measurement and Command 129 reads the signal from the field device. Once that is complete, the signal is displayed to the user. The user may take another reading if this one is unsatisfactory.

Once an interesting reading is found it can be added to the "stored_signals" FILE, replace an existing signal stored in "stored_signals", or compare the current reading to a stored signal. In many cases, the stored signals are stepped through as in the previous example.

In the insertion case, the user may insert the new data at the front, back, or in the middle of the LIST contained in the FILE "stored_signals". This section of code shows the use of the COUNT attribute and the ListInsert() Builtin function.

The section performing the replace function orders the list to find the element to be replaced. A call to the ListDeleteElementAt() Builtin followed by a ListInsert().call affects the replacements.

In the final section of the example, the current radar signal is compared to a stored signal. The list is stepped through and the "compare_radar_signals" GRAPH displays both signals on the same graph.

```

VARIABLE ping_index      { TYPE INDEX ping_data; }
ARRAY    ping_data      { TYPE sample; NUMBER_OF_ELEMENTS 4096; }
VARIABLE today           { TYPE DATE; }
VARIABLE user            { TYPE ASCII(32); }
VARIABLE conditions      { TYPE ASCII(64); }

COLLECTION liveSig
{
    MEMBERS
    {
        DATE_STAMP, today;
        TAKEN_BY, user;
        NOTES, conditions;
        DT, sample_interval;
        SIGNAL, pind_data;
    }
}

COMMAND ping
{
    NUMBER 128;
    OPERATION COMMAND;

    TRANSACTION
    {
        REQUEST { }
        REPLY { response_code, device_status }
    }
    RESPONSE_CODES { ... }
}

COMMAND read_ping_data
{
    NUMBER 129;
    OPERATION READ;

    TRANSACTION
    {
        REQUEST { ping_index (INFO, INDEX) }
        REPLY
        {
            response_code, device_status, ping_index,
            ping_data[ping_index+00], ping_data[ping_index+01],
            ping_data[ping_index+02], ping_data[ping_index+03],
            ping_data[ping_index+04], ping_data[ping_index+05],
            ping_data[ping_index+06], ping_data[ping_index+07],
            ping_data[ping_index+08], ping_data[ping_index+09],
            ping_data[ping_index+10], ping_data[ping_index+11],
            ping_data[ping_index+12], ping_data[ping_index+13],
            ping_data[ping_index+14], ping_data[ping_index+15]
        }
    }
    RESPONSE_CODES { ... }
}

WAVEFORM new_echo
{
    TYPE YT
    {
        X_INITIAL 0;
        X_INCREMENT sample_interval;
        Y_VALUES { echo_signal }
    }
}

GRAPH new_radar_signal
{
    MEMBERS
    {
        PING, new_echo;
    }
}

GRAPH compare_radar_signals
{
    MEMBERS
    {
        PING1, new_echo;
        PING2, one_echo;
    }
}

```

```

}

/*
*****
* pings the radar and captures the echo waveform into ping_data
*/
#define PING_COMPLETE 0x10;

METHOD newPing
{
    LABEL "Ping";
    DEFINITION
    {
        int i,
            ans;          /*The users answer to select from list*/
        long selector;
        long varid[5];     /*used in the acknowledge Builtin calls*/

        i = 0;

        varid[0] = VARID( date_taken );
        varid[1] = VARID( operator );
        varid[2] = VARID( operating_conditions );
        varid[3] = VARID( sample_interval );
        varid[4] = VARID( echo_signal );

        /*
        * ping once to get a radar signal
        */
        do {
            select_from_list ("Ready to make a measurement","Yes;No",ans);
            while (ans == 0) {
                send(128);          /* ping */
                do {                /* check the status of the ping */
                    send(48);
                } while(GET_VAR_VALUE (xmtr_specific_status4) & PING_COMPLETE);
            }
            /*
            * ping complete, read the signal
            */
            for (ping_index = 0; ping_index < 4096; ping_index += 16) {
                send (129);
            }
            MenuDisplay (new_radar_signal_menu, "OK", selector);
            /*
            * assumes the graph is displayed until I destroy it
            */
            select_from_list ("Take more radar data","Yes;No",ans);
        }

        /*
        * radar signal looks good. save it?
        */
        select_from_list("what now?", "save the signal;" \
                        "replace a stored signal; compare signals;" \
                        "get new signal; quit", ans);

        switch (ans)
        {

            case 0:          /* save the signal */
                get_dev_var_value (conditions);
                select_from_list("save where?","front of list; end of list;
                                insert into list", ans);

                switch (ans) {
                    case 0: i = 0; break;          /* beginning */
                    case 1: i = recorded_signals.COUNT; break; /* end */

                    default: /* insert */
                        i = 0;
                        do {
                            /* get the current record so we can display it */
                            operator = stored_signals.SIGNALS[i].TAKE_BY;
                            operating_conditions =
                                stored_signals.SIGNALS[i].NOTES;
                            sample_interval = stored_signals.SIGNALS[i].DT;
                            echo_signal = stored_signals.SIGNALS[i].SIGNAL;

                            /* display the graph and the annotations. */
                            MenuDisplay (review_radar_signal_menu,

```

```

"OK", selector);
        acknowledge ( "Radar Signal by %{1} \n" \
            "Description %{2}" varid);

        select_from_list ("Insert here?", "Yes;No", ans);

        if (ans == 0) {
            break;
        }
        i++;
    } while ( i < recorded_signals.COUNT);
    break;

}
ListInsert ( storedSignals.SIGNALST, i, liveSig );
break;

case 1:          /* replace a stored signal */
    i = 0;
    do {
        /* get the current record so we can display it */
        operator          = stored_signals.SIGNALS[i].TAKE_BY;
        operating_conditions= stored_signals.SIGNALS[i].NOTES;
        sample_interval    = stored_signals.SIGNALS[i].DT;
        echo_signal        = stored_signals.SIGNALS[i].SIGNAL;

        /* display the graph and the annotations. */

        MenuDisplay (review_radar_signal_menu, "OK", selector);
        acknowledge ( "Radar Signal by %{1} \n" \
            "Description %{2}" varid);
        select_from_list ("replace signal?", "Yes;No", ans);

        if (ans == 0) {
            ListDeleteElement ( storedSignals.SIGNALST, i );
            ListInsert ( storedSignals.SIGNALST, i, liveSig );
            break;
        }
        i++;
    } while ( i < recorded_signals.COUNT);
    acknowledge("no signals replaced");
    break;

case 2:          /* compare signals */
    i = 0;
    do {
        /* get the current record so we can display it */
        operator          = stored_signals.SIGNALS[i].TAKE_BY;
        operating_conditions= stored_signals.SIGNALS[i].NOTES;
        sample_interval    = stored_signals.SIGNALS[i].DT;
        echo_signal        = stored_signals.SIGNALS[i].SIGNAL;

        /* display the graph and the annotations. */

        MenuDisplay (review_radar_signal_menu, "OK", selector);
        acknowledge ( "Radar Signal by %{1} \n" \
            "Description %{2}" varid);
        select_from_list ("compare with this signal?",

            "Yes;No", ans);

        if (ans == 0) {
            MenuDisplay ( compare_radar_signal_menu, "OK", selector);
            select_from_list ("compare with another
                signal?", "Yes;No", ans);

            if (ans==0) {
                continue;
            }
            break;
        }
        i++;
    } while ( i < recorded_signals.COUNT);
    acknowledge("no signals compared");
    break;

case 3:          /* get new signal */
    continue;

```

```

        default:      /* quit */
            process_abort ();
            break;
    }
} while ( 1 );
}

```

Figure 62 – Example of an EDD that inserts, replaces, or compares radar signals

5.3 Exposing data items outside the EDD application

Any data items with VALIDITY equal to FALSE shall not be exposed.

The EDD application shall not expose data items outside the EDD application, if the PRIVATE attribute is evaluated to TRUE. For a COLLECTION, RECORD or a REFERENCE_ARRAY, where the PRIVATE attribute is evaluated to FALSE or it is not defined, the EDD application shall expose the COLLECTION, RECORD or the REFERENCE_ARRAY and all referenced items, regardless of the PRIVATE attribute of the referenced data items.

For a COLLECTION, RECORD or a REFERENCE_ARRAY, where the PRIVATE attribute is evaluated to TRUE, the COLLECTION, RECORD or REFERENCE_ARRAY itself is not exposed, but the referenced data items may be exposed by means of other rules.

5.4 Initialization of EDD instances

5.4.1 Overview

Initialization of variables is important for offline configuration. If a device is configured, e.g. in the planning phase, the configuration may be made consistent with the devices. The EDD developer defines ranges and enumerations including conditionals to allow the EDD application to check the consistency and avoid problems while downloading the configuration.

5.4.2 Initialization support

If the user selects a device then an instance is created by the EDD application with the related EDD. The EDD variable values shall be initialized using the following rules.

- a) Variables initialized with the variable type initial value. This shall apply if b), c) or d) do not apply.
- b) Variable initialization with EDDL INITIAL_VALUES. This shall apply if a) does not apply.
- c) Variable initialization with DEFAULT_VALUES. This shall apply if INITIAL_VALUES do not exist.
- d) Variable initialization using EDDL COMPONENTS.

In addition the user may select a TEMPLATE that contains further initializations. FOUNDATION fieldbus, PROFIBUS and PROFINET may have additional files to the EDD that may define default values.

5.4.3 TEMPLATE

TEMPLATES specify default parameter values for different uses of a device, i.e., they are application specific. TEMPLATES may be specified by the device manufacturer in their EDD. For example, a differential pressure device may be configured as a difference presser sensor for a heat exchanger or as a level sensor in a hydrostatic level application. TEMPLATES may also target specific market segments. For example, pharmaceutical requirements of traceability and validation may affect more parameters than may be required by other markets such as factory automation. There may be more than one TEMPLATE for a given device.

An EDD may include templates, as specified in IEC 61804-3. Figure 63 is an example of a TEMPLATE.

```

// Template for differential pressure
TEMPLATE diff_pressure_template
{
    HELP [diff_pressure_help];
    LABEL "Differential pressure";
    DEFAULT_VALUES
    {
        variable-reference = constant-expression;
        variable-reference = constant-expression;
        ...
    }
}
// Template for absolute pressure
TEMPLATE abs_pressure_template
{
    HELP [abs_pressure_help];
    LABEL "Absolute pressure";
    DEFAULT_VALUES
    {
        variable-reference = constant-expression;
        variable-reference = constant-expression;
        ...
    }
}

```

Figure 63 – Example of TEMPLATE usage

The EDD application shall provide a list of all templates defined in the EDD to the user. If the user select a template at any time, the EDD application shall reinitialize all in the template specified VARIABLES with the define value. The EDD application shall allow the user to apply different or the same template multiple times, the last value change remains.

Initialization of a referenced VARIABLE from a VALUE_ARRAY or LIST TYPE shall not effect array or list elements, it change only the direct referenced VARIABLE.

5.5 Device model mapping

5.5.1 BLOCK_A

A FOUNDATION fieldbus EDD may contain device level root menus (e.g. process_variables_root_menu) or block level menus (e.g. process_variables_root_menu_ai).

All MENU ITEMS specified in a device-level menu shall explicitly specify the block with which they are associated.

Block level menus shall be referenced in the BLOCK_A MENU_ITEMS attribute. Block-level menus shall not include menu items that reference a specific block. For example, this means a menu or method that contains a conditional VALIDITY that in turn contains a reference to a specific block may not be placed on a block-level menu, either directly or indirectly.

The example Figure 64 illustrates referencing according to the correct context.

```

BLOCK __analog_input_block
{
    CHARACTERISTICS __ai_character;
    LABEL [analog_input_block];
    HELP [analog_input_block_help];
    PARAMETERS
    {
        ST_REV, __st_rev;
        TAG_DESC, __tag_desc;
        /* ... */
    }
    MENU_ITEMS
    {
        process_variable_root_menu_ai; /* block based menu */
    }
}

MENU process_variable_root_menu /* device level menu */
{
    ITEMS
    {
        __analog_input_block[0].PARAM.ST_REV; /* block reference specifying the block instance */
    }
}

MENU process_variable_root_menu_ai /* a Block level menu */
{
    ITEMS
    {
        PARAM.ST_REV; /* menu is associated with a block type, */
                     /* so only parameter referencing is used */
    }
}

```

Figure 64 – Example of a BLOCK_A

5.5.2 BLOCK_B

EDDs for PROFIBUS and PROFINET devices shall contain all necessary device level menus. BLOCK_B level menus do not exist for PROFIBUS and PROFINET devices.

The BLOCK_B definition is only used to address device data of PI Profile for Process Control Devices (see 10.1.4.3).

6 EDDL METHOD programming and usage of Builtins

6.1 Builtin MenuDisplay

The Builtin MenuDisplay is modelled very closely to the Builtin select_from_list/select_from_menu that provides users with the ability to select from a set of choices in a method. Instead of displaying text strings, the Builtin MenuDisplay will display the specified menu and append the user selected choices as buttons (or equivalent). Once the user has selected the option, the menu is dismissed and control is returned to the method.

The user selection is returned through the selection parameter based on the position of the element in the options list. A semicolon delimits choices. It is recommended that the selection list contain three or fewer entries.

The value returned by the selection is zero-based. For example, if the option list contains "BACK;NEXT", selecting BACK would return zero and selecting NEXT would return a one. If selection is an empty string, the EDD application will append a default button and return zero if selected by the user.

If the EDD application is unable to display the menu, an error status is returned by the Builtin.

In addition to the specified selection items, the EDD application should also provide a cancel button (or equivalent) that will force the dismissed menu and invoke the method's abort routines.

Menus that are called using the MenuDisplay are of type DIALOG or WINDOW. Methods and edit displays are not permitted menu items (including referenced submenus, if any appears on a menu called from MenuDisplay).

NOTE PROFIBUS and PROFINET support nesting methods with UI Builtins.

Figure 65 shows an EDD example of a three-step set-up wizard.

Withdrawing
IECNORM.COM: Click to view the full PDF of IEC 61804-4:2015

```

IMAGE deviceimage
{
    PATH "device_image.jpg"
}

MENU wizard_step0
{
    LABEL "Wizard Step 1";
    STYLE DIALOG;
    ITEMS
    {
        "Welcome to the device setup wizard. ",
        deviceimage
    }
}

MENU wizard_step1
{
    LABEL "Wizard Step 2";
    STYLE DIALOG;
    ITEMS
    {
        "Enter the setup values",
        devicevar1,
        devicevar2
    }
}

MENU wizard_step2
{
    LABEL "Wizard Step 3";
    STYLE DIALOG;
    ITEMS
    {
        "Wizard Complete",
    }
}

METHOD setup_wizard
{
    CLASS INPUT;
    LABEL "Device Setup Wizard";
    DEFINITION
    {
        long select=0;
        long step=0;
        do
        {
            switch (step)
            {
                case 0:
                    MenuDisplay(wizard_step0, "NEXT", select);
                    step++;
                    break;
                case 1:
                    MenuDisplay(wizard_step1, "BACK;NEXT", select);
                    if (select==0) step=0;
                    if (select==1) step=2;
                    break;
                case 2:
                    MenuDisplay(wizard_step2, "FINISH", select);
                    step++;
                    break;
            }
        } while (step<3);
    }
}

```

Figure 65 – Example of a wizard

6.2 Division by zero and undetermined floating values

6.2.1 Integer and unsigned integer values

The EDD application shall raise a runtime exception, if the evaluation of an expression contains an integer division by zero or modulo calculation to zero.

If the exception arises inside a method execution, the EDD application shall abort the method.

If the exception arises at an expression or conditional evaluation, the EDD application shall stop execution of the EDD and notify the user.

6.2.2 Floating-point values

The EDD application shall be able to handle division by zero for floating-point operations as defined in IEEE 754. The rules shall be applied inside method execution and in expression evaluation of attributes.

Table 9 shows examples of results of floating-point calculations, whereas 'n' is a positive numeric value.

Table 9 – Examples of floating-point results

Expression	Result
1.0/0.0	Infinity
-1.0/0.0	-Infinity
n / +(-)Infinity	0
+(-)Infinity x +(-)Infinity	+(-)Infinity
+(-)nonzero / 0	+(-)Infinity
Infinity + Infinity	Infinity
0.0/0.0	NaN
Infinity - Infinity	NaN
+(-)Infinity / +(-)Infinity	NaN
+(-)Infinity * 0	NaN
sqrt (-n)	NaN
log (-n)	NaN
log10 (-n)	NaN
log2 (-n)	NaN
asin (<-1), asin(>1)	NaN
acos (<-1), acos(>1)	NaN

The representation of these three results may differ in the EDD Applications. NaN shall be displayed as "NaN" or "Not a Number", Infinity as "Inf" or "Infinity". The algebraic sign of infinity shall also be displayed.

7 Modular devices

7.1 Overview

The intention of the modular device concept is to provide a means to reduce the overall size and complexity of an individual EDD through the reduction of conditional statements. This is accomplished by creating EDDs for each component of the modular device. This concept allows for the component EDD to be delivered independently. The configuration description of modular devices includes information about allowed components and their relationships.

A component is a piece of software or hardware that must be contained within the modular device, i.e. a module can not function separately from the modular device hosting it. A component may support one or more modular devices. Components may contain additional components e.g. channels, but consideration should be given to user deployment. Additional EDDs may create problems for users as they upgrade their systems with new EDDs.

7.2 EDD identification

Two identification techniques exists for EDDs.

- a) EDD reference: EDDs are identified for each protocol by MANUFACTURER, DEVICE_TYPE and DEVICE_REVISION. The referencing allows to identify one EDD.
- b) EDD referencing using COMPONENT description: EDDs are identified by PROTOCOL, CLASSIFICATION, MANUFACTURER and COMPONENT_PATH. This referencing allows referencing one or many EDDs. To reference more than one EDD, for example any pressure devices, only the classification could be specified.

The referencing between the components that describes allowed relations is handled through the COMPONENT_REFERENCE. A COMPONENT_REFERENCE identifies an EDD in the EDD library.

The attribute COMPONENT_PATH has different usages depending on the construct, see Table 10.

Table 10 – Usages of COMPONENT_PATH

Construct	Usage
COMPONENT	Creates a component in the catalog hierarchy
COMPONENT_FOLDER	Creates a folder in the catalog hierarchy
COMPONENT_REFERENCE	In this use case the attribute does not affect the catalog. It is used only to build the reference

The COMPONENT_PATH is always relative to the COMPONENT_PARENT, if defined. Absolute paths (e.g. "/PROTOCOL/..") are not permitted.

The PROTOCOL, CLASSIFICATION and MANUFACTURER are defined in IEC 61804-3. The manufacturer further defines the hierarchy through the catalog path referenced from the above.

7.3 Instance object model

Modular devices consist of software and hardware modules. Each module can be described in a separate EDD. The EDDs include the configuration description. The modular device or modules EDD can be delivered independently at different times. The configuration description of modular devices includes information about allowed module types and the allowed number of modules. This configuration description allows an EDD application to configure a modular device in a hierarchy, see Figure 66.

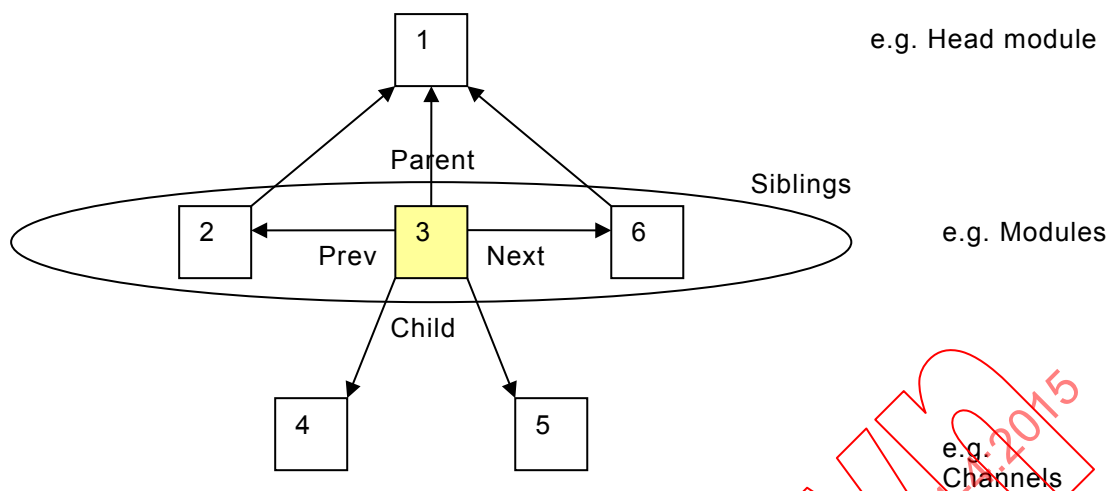


Figure 66 – The different relations of a module

7.4 Offline configuration

The component description allows the EDD application to guide the user through the configuration of modular devices. The topologies will be restricted by the defined component types and the number of components. An EDD application can provide a list of component types that can be instantiated for a modular device during offline configuration. The topology of a modular device should be checked by the EDD application during configuration and before downloading/uploading data.

7.5 Online configuration

The device configuration is read from the device during commissioning or runtime. This can be done through protocol conventions or through SCAN and DETECT methods, see 7.10.

7.6 Simple modular device example

7.6.1 General

In the examples in Figure 67 a simple modular device is described that allows plugging two types of modules into two slots.

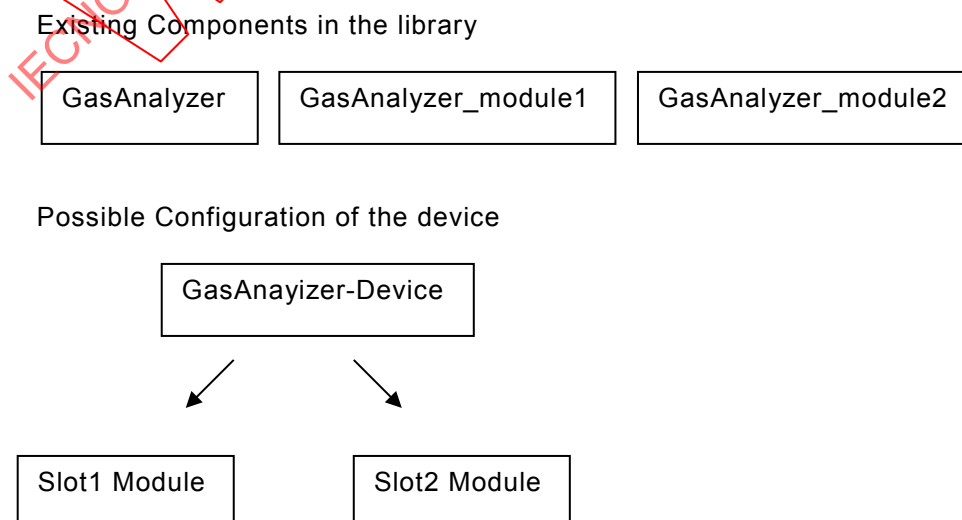


Figure 67 – Components and possible configuration of the modular devices

Three EDD examples are given in 7.6 showing the simple modular device.

- a) Separate EDD file example with direct EDD referencing, see 7.6.2.
- b) Separate EDD file example with catalog EDD referencing, see 7.6.3.
- c) One EDD file example, see 7.6.4.

7.6.2 Separate EDD file example with direct EDD referencing

Figure 68 shows an EDD example for a modular device.

```

MANUFACTURER 0x10ff79,
DEVICE_TYPE 0x1000,
DEVICE_REVISION 0x01,
DD_REVISION 0x01

COMPONENT GasAnalyzer
{
    LABEL "GasAnalyzer";
    COMPONENT_RELATIONS { GasAnalyzer_module_relation }
}

COMPONENT_RELATION GasAnalyzer_module_relation
{
    LABEL "GasAnalyzer Module";
    RELATION_TYPE CHILD_COMPONENT;

    COMPONENTS { GasAnalyzer_module1}

    MINIMUM_NUMBER 0;
    MAXIMUM_NUMBER 2;
}

COMPONENT_REFERENCE GasAnalyzer_module1
{
    DEVICE_TYPE 0x1001;
    DEVICE_REVISION 0x01;
}

VARIABLE device_mode
{
    LABEL "Mode";
    CLASS CONTAINED;
    TYPE ENUMERATED
    {
        {1, "simple"},
        {2, "complex"}
    }
}

MENU root_menu
{
    LABEL "Main Menu";
    ITEMS
    {
        device_mode,
        GasAnalyzer_module_relation[0].module_type,
        GasAnalyzer_module_relation[1].module_type
    }
}

```

Figure 68 – Separate EDD file example with direct EDD referencing

Figure 69 shows the EDD example for module1.

```

MANUFACTURER 0x10ff79,
DEVICE_TYPE 0x1001,
DEVICE_REVISION 0x01,
DD_REVISION 0x01

COMPONENT GasAnalyzer_module1
{
    LABEL "GasAnalyzer Module 1";
}

VARIABLE module_type
{
    LABEL "Module type";
    TYPE ENUMERATED
    {
        DEFAULT VALUE 1;
        {1, "GasAnalyzer module 1"},
        {2, "GasAnalyzer module 2"}
    }
}

```

Figure 69 – EDD example for module1

Figure 70 shows the EDD example for module2. The relation to that component is not described in the modular device EDD.

```

MANUFACTURER 0x10ff79,
DEVICE_TYPE 0x1002,
DEVICE_REVISION 0x01,
DD_REVISION 0x01

COMPONENT GasAnalyzer_module2
{
    LABEL "GasAnalyzer Module 2";
    COMPONENT_RELATIONS { GasAnalyzer_parent_relation }
}

COMPONENT_RELATION GasAnalyzer_parent_relation
{
    RELATION_TYPE PARENT_COMPONENT;
    COMPONENTS { GasAnalyzer }
}

COMPONENT_REFERENCE GasAnalyzer
{
    DEVICE_TYPE 0x1000
    DEVICE_REVISION 0x01
}

VARIABLE module_type
{
    LABEL "Module type";
    TYPE ENUMERATED
    {
        DEFAULT VALUE 2;
        {1, "GasAnalyzer module 1"},
        {2, "GasAnalyzer module 2"}
    }
}

```

Figure 70 – EDD example for module2

7.6.3 Separate EDD file example with classification EDD referencing and interfaces

Figure 71 shows an EDD example for a modular device.

```

MANUFACTURER 0x10ff79,
DEVICE_TYPE 0x1000,
DEVICE_REVISION 0x01,
DD_REVISION 0x01

#include "module_interface.dd"
#include "GasAnalyzer_interface.dd"

COMPONENT GasAnalyzer
{
    LABEL                "GasAnalyzer";
    PROTOCOL              PROFIBUS_DP;
    CLASSIFICATION        SENSOR_ANALYTIC;
    COMPONENT_PATH        "GASANALYZER";

    COMPONENT_RELATIONS  { GasAnalyzer_module_relation }
    SUPPLIED_INTERFACE    { GasAnalyzer_interface }
}

COMPONENT_RELATION GasAnalyzer_module_relation
{
    LABEL                "GasAnalyzer_module_relation";
    RELATION_TYPE         CHILD_COMPONENT;

    COMPONENTS            { GasAnalyzer_module1}

    REQUIRED_INTERFACE     { module_interface }
    SUPPLIED_INTERFACE    { GasAnalyzer_interface }

    MINIMUM_NUMBER        0;
    MAXIMUM_NUMBER        2;
}

COMPONENT_REFERENCE GasAnalyzer_module1
{
    PROTOCOL              PROFIBUS_DP;
    CLASSIFICATION        SENSOR_ANALYTIC;
    COMPONENT_PATH        "GASANALYZER/MODULE1";
}

MENU    root_menu
{
    LABEL    "Main Menu";
    ITEMS
    {
        device_model
        GasAnalyzer_module_relation[0].module_interface.module_type;
        GasAnalyzer_module_relation[1].module_interface.module_type;
    }
}

```

Figure 71 – EDD example for modular device

Figure 72 shows an EDD example for module1, a component supported by that modular device.

```
MANUFACTURER 0x10ff79,
DEVICE_TYPE 0x1001,
DEVICE_REVISION 0x01,
DD_REVISION 0x01

#include "module_interface.dd"

COMPONENT GasAnalyzer_module1
{
    LABEL                "GasAnalyzer Module 1";
    PROTOCOL              PROFIBUS_DP;
    CLASSIFICATION        SENSOR_ANALYTIC;
    COMPONENT_PATH        "GASANALYZER/MODULE1";

    SUPPLIED_INTERFACE    { module_interface }

    INITIAL_VALUES
    {
        module_type      = 1;
    }
}
```

Figure 72 – EDD example for module1

Figure 73 shows an EDD example for module2. The relation to that component is not described in the modular device EDD.

```
MANUFACTURER 0x10ff79,
DEVICE_TYPE 0x1002,
DEVICE_REVISION 0x01,
DD_REVISION 0x01

#include "module_interface.dd"

COMPONENT GasAnalyzer_module2
{
    LABEL                "GasAnalyzer Module 2";
    PROTOCOL              PROFIBUS;
    CLASSIFICATION        SENSOR_ANALYTIC;
    COMPONENT_PATH        "GASANALYZER/MODULE2";

    COMPONENT_RELATIONS  { GasAnalyzer_parent_relation }
    SUPPLIED_INTERFACE    { module_interface }

    INITIAL_VALUES
    {
        module_type      = 2;
    }
}

COMPONENT_RELATION GasAnalyzer_parent_relation
{
    RELATION_TYPE          PARENT_COMPONENT;
    COMPONENTS              { GasAnalyzer }
    SUPPLIED_INTERFACE      { module_interface }
}

COMPONENT_REFERENCE GasAnalyzer
{
    PROTOCOL                PROFIBUS_DP;
    CLASSIFICATION           SENSOR_ANALYTIC;
    COMPONENT_PATH           "GASANALYZER";
}
```

Figure 73 – EDD example for module2

7.6.4 One EDD file example

The modular device and the two modules types are described within one EDD source file (see Figure 74).

```
MANUFACTURER 0x10ff79,
DEVICE_TYPE 0x1000,
DEVICE_REVISION 0x01,
DD_REVISION 0x01

COMPONENT GasAnalyzer
{
    LABEL "GasAnalyzer";
    COMPONENT_RELATIONS { GasAnalyzer_module_relation }
}

COMPONENT_RELATION GasAnalyzer_module_relation
{
    RELATION_TYPE CHILD_COMPONENT;
    COMPONENTS { GasAnalyzer_module1, GasAnalyzer_module2 }
    MINIMUM_NUMBER 0;
    MAXIMUM_NUMBER 2;
}

COMPONENT GasAnalyzer_module1
{
    LABEL "GasAnalyzer Module 1";

    DECLARATION
    {
        /* Any device type specific declaration may follow here */
        VARIABLE module_type
        {
            LABEL "Module type";
            TYPE ENUMERATED
            {
                DEFAULT_VALUE 1;
                {1, "GasAnalyzer module 1"},
                {2, "GasAnalyzer module 2"}
            }
        }
    }
}

COMPONENT GasAnalyzer_module2
{
    LABEL "GasAnalyzer Module 2";

    DECLARATION
    {
        /* Any device type specific declaration may follow here */
        VARIABLE module_type
        {
            LABEL "Module type";
            TYPE ENUMERATED
            {
                DEFAULT_VALUE 2;
                {1, "GasAnalyzer module 1"},
                {2, "GasAnalyzer module 2"}
            }
        }
    }
}

VARIABLE device_mode
{
    LABEL "Mode";
    TYPE ENUMERATED
    {
        {1, "simple"},
        {2, "complex"}
    }
}

MENU root_menu
{
    LABEL "Main Menu";
    ITEMS
    {
```

```

        device_mode,
        GasAnalyzer_module_relation[0].module_type;
        GasAnalyzer_module_relation[1].module_type;
    }
}

```

Figure 74 – EDD example for module2

7.6.5 Combination of single and separate modular device example

The techniques to specify components in one or more files can be combined, for example the simple file example could be extended with a module3 that is described in a separate file.

7.7 COMPONENT_RELATION

7.7.1 General

A COMPONENT_RELATION defines the required and supplied INTERFACES for a special kind of component subcomponent relationship.

The specification of COMPONENT can be omitted if the child COMPONENT describes the relation using an appropriate PARENT_RELATION.

7.7.2 NEXT_COMPONENT usage

The example in Figure 75 shows that the next “slot” of a module has to be empty by specifying an empty COMPONENTS list. This COMPONENT_RELATION shall be defined in the EDD of the module where the next slot should be empty.

```

COMPONENT_RELATION GasAnalyzer_module_relation
{
    RELATION_TYPE      NEXT_COMPONENT;
    ADDRESSING          { slot } // list of addressing variables

    COMPONENTS          { }
}

```

Figure 75 – NEXT_COMPONENT usage

7.7.3 REQUIRED_RANGES and ADDRESSING usage

The example in Figure 76 shows a module that is only allowed to plug in “slots” 0 to 4. “slot” is a variable of GasAnalyzer_module1 that is also used for addressing. The module2 can be plugged in any slot.

```

COMPONENT_RELATION GasAnalyzer_module_relation
{
    RELATION_TYPE      CHILD_COMPONENT;
    ADDRESSING          { slot }

    COMPONENTS          {
        GasAnalyzer_module1 {REQUIRED_RANGES {slot{MIN_VALUE 0; MAX_VALUE 4;}}},
        GasAnalyzer_module2
    }

    MINIMUM_NUMBER      0;
    MAXIMUM_NUMBER      10;
}

```

Figure 76 – REQUIRED_RANGES usage

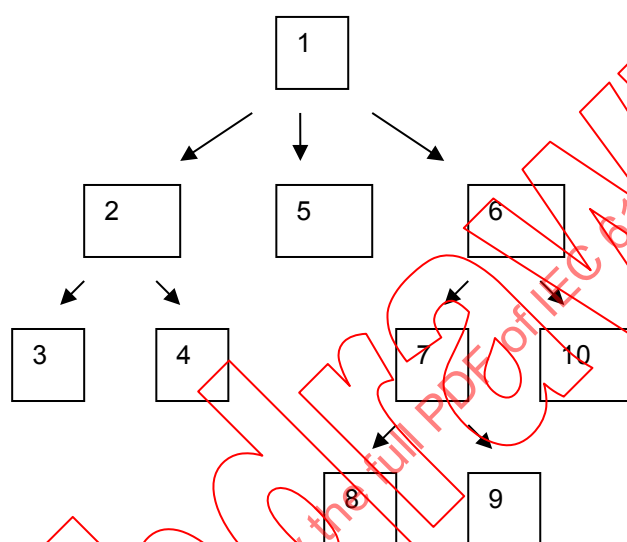
7.8 Upload and download for modular devices

Modular devices can support upload or/and download for the different parts of the modular device. Each EDD can support the upload/download mechanism by using the corresponding menu definitions (see the upload/download section in the offline configuration). The upload/download menus can reference variables and menus from underlying components (CHILD_COMPONENTS).

If a module does not support the upload or download because e.g. it only has volatile memory, the EDD should define the upload or download menu with no variables in the items and a static text e.g. “n.a.”.

The EDD application starts upload or download at the parent followed by the first child, the next to the child and so on. If any child by itself has children then these children are performed before the next sibling is uploaded or downloaded.

If an ADDRESSING is defined in the COMPONENT_RELATION then the ordering of the children is defined by the ascending addresses, see Figure 77.



NOTE The arrows show the hierarchy, the numbers show the ordering.

Figure 77 – Upload/download order of a modular device

If an error occurs while processing an upload or download, the EDD application shall not try to proceed to the children of the module that produced the error.

The EDD application may allow the user to download or upload parts of a modular devices.

7.9 Diagnostic

Each module should support diagnostic by implementing a diagnostic EDD method that is additional to the module specific diagnostic.

The diagnostic classification should be extended with two additional cases.

Additional diagnostic classifications are shown in Table 11.

Table 11 – Diagnostic classifications

Classification	Description
Type mismatch	The connected device or module is not of the projected type. In case of type mismatch any other diagnostic checks in the diagnostic METHOD may not be possible. The EDD application should not force any communication until projected device type is equal to the connected device. In case of type mismatch, the EDD application can invoke the detect METHOD to get the right type (EDD).
Different device instance	Connected device or module has the projected type but it is a different instance than stored in the configuration data base
NOTE 1 This is only needed if no common way of EDD identification exists, e.g. for PROFIBUS.	
NOTE 2 Inconsistency between the device configuration read from the device (online) and the offline dataset can exist due to several use cases. Examples are: module replacement, incorrect installation or wiring, incorrect configuration in offline database.	

7.10 Reading modular device topology

7.10.1 SCAN

With an optional SCAN EDD METHOD an EDD application is able to get a list of existing sub-components. The SCAN METHOD reads the device information to create a list of the contained components (SCAN_LIST). The EDD application uses this list to create instances for the components.

Figure 78 shows an example of a device SCAN METHOD.

```

...
COMPONENT MyDevice
{
    ...
    SCAN          MyScan;           // makes the scan method public
    SCAN_LIST     MyScanList;       // makes the scan list public
}

VARIABLE Relation { ... TYPE ASCII(64) ... }
VARIABLE Type { ... TYPE ASCII(64) ... }
VARIABLE Address { ... TYPE INTEGER(4) ... }

COLLECTION ComponentInfo
{
    MEMBER
    {
        RELATION, Relation; // describes how the sub-component is used
        TYPE, Type; // reference to an EDD in the EDD library
        ADDRESS, Address; // address of found sub-component
    }
}

LIST MyScanList
{
    ...
    TYPE ComponentInfo;
}

METHOD MyScan
{
    TYPE int; // the method returns the number of children found or FAILURE
    DEFINITION
    {
        int i;
        while ( MyScanList.COUNT > 0 ) // delete entire previous list content
            ListDeleteElementAt(MyScanList,0);

        ... // read information about available modules
        ComponentInfo.Relation = "Modules";
        i = 0;
        while( ... ) // no error and more info available
        {
            ... // read module information
            ComponentInfo.Type = "99,12,1"; // manufacturer, device type,
            // device revision
            ComponentInfo.Address = ...; // address
            ListInsert ( MyScanList, i, ComponentInfo );

            i++;
        }

        if ( ... ) // if no error
            return i;

        return FAILURE;
    }
}

```

Figure 78 – Example of a SCAN METHOD

The SCAN_LIST is a LIST of COLLECTIONS. The COLLECTION shall contain the COMPONENT_RELATION name, the type and all address information defined in ADDRESSING. The result of the SCAN METHOD can be a specific EDD or a group of EDDs. In case of specific EDDs, the EDD application does not need to call detect the METHODS. In case of a group of EDDs, the type may contain the COMPONENT_PATH.

The EDD application uses the COMPONENT_RELATION name and the EDD to know how to create instances with this information.

7.10.2 Detect module type

If the SCAN METHOD has identified an EDD of a component family, then the DETECT METHOD of this EDD can be used to identify the EDD for the device.

The DETECT METHOD reads all necessary information of the device to identify the sub-component. The result of the DETECT METHOD is the identification of an EDD.

In case that the DETECT METHOD can not identify the component EDD, it can return the identification of another component family EDD.

Figure 79 shows an example of a device DETECT METHOD.

```
...
COMPONENT MyDevice
{
    ...
    DETECT      MyDetect;
}

METHOD MyDetect ( DD_STRING &MyComponentType )
{
    TYPE int;      // returns SUCCESS if device is known, FAILURE if device is unknown or
                  // detection is not possible e.g. communication problems.
    DEFINITION
    {
        ...      // reads device information to identify the device
        if ( ... ) // if device is known
        {
            MyComponentType = "99,19,3";
            return SUCCESS;
        }
        return FAILURE;
    }
}
```

Figure 79 – Example of a DETECT METHOD

The following rules apply for SCAN and DETECT.

- SCAN and DETECT METHODS should not use any user interface Builtins.
- If the DETECT METHOD does not detect a specific EDD, then the EDD application should invoke the DETECT METHOD of the family EDD.

7.11 Configuration check

The CHECK_CONFIGURATION METHOD checks the offline configuration of a device. If any warnings or errors are detected it returns a list of messages that the EDD application should display to the user. If no warning or error occurs, the METHOD shall return BI_SUCCESS and may not change the MessageList.

Figure 80 shows an example of the usage of CHECK_CONFIGURATION METHOD.

```

VARIABLE ConfigCheckMessage { ... TYPE ASCII(256); }
LIST ConfigCheckMessageList { ... TYPE ConfigCheckMessage; }

COMPONENT MyDevice
{
    CHECK_CONFIGURATION          MyConfigCheck;
}

METHOD MyConfigCheck ( DD_STRING &MessageList )
{
    TYPE int; // BI_SUCCES configuration is valid,
              // BI_ERROR configuration is invalid
    DEFINITION
    {
        if (...) // check of the configuration
        {
            MessageList = "Configuration error ...";
            return BI_ERROR;
        }
        else
            return BI_SUCCESS;
    }
}

```

Figure 80 – Example of a CHECK_CONFIGURATION METHOD

NOTE If the CHECK_CONFIGURATION METHOD is written in a way that allows using the method in offline and online, then the method can be called in the diagnostic method to check the device configuration as part of the whole diagnostic check.

8 Edit session

8.1 Data management

8.1.1 Overview

The session management defines the data handling for dialogs, windows and methods. Each dialog, window or method shall have its own scope of the data represented by a cache.

A cache contains the changed variable values to separate the data modifications of a session. When the session is confirmed, the changes shall be made persistent.

The EDD application shall use separate caches for offline and online sessions. For offline sessions the EDD application shall use an offline cache with the values of CLASS LOCAL and non-LOCAL variables. The EDD application may store the offline cache persistently.

Figure 81 shows the data caching for an offline session.

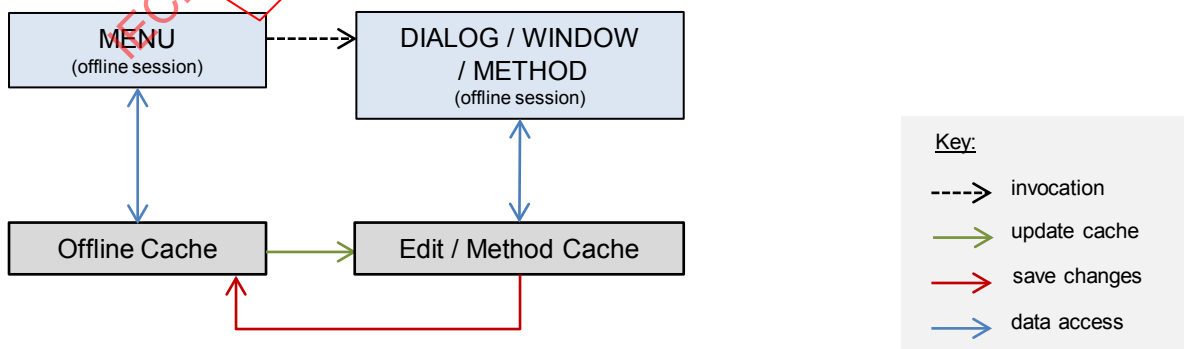


Figure 81 – Data caching for an offline session

For online sessions the EDD application shall use an online cache. The values of the requested CLASS LOCAL variables shall be retrieved from the offline cache for PROFIBUS

and PROFINET devices; otherwise the values of the requested CLASS LOCAL variables shall be initialized with DEFAULT_VALUES. The values of the requested non-LOCAL variables shall be read from the device.

Figure 82 shows the data caching for an online session.

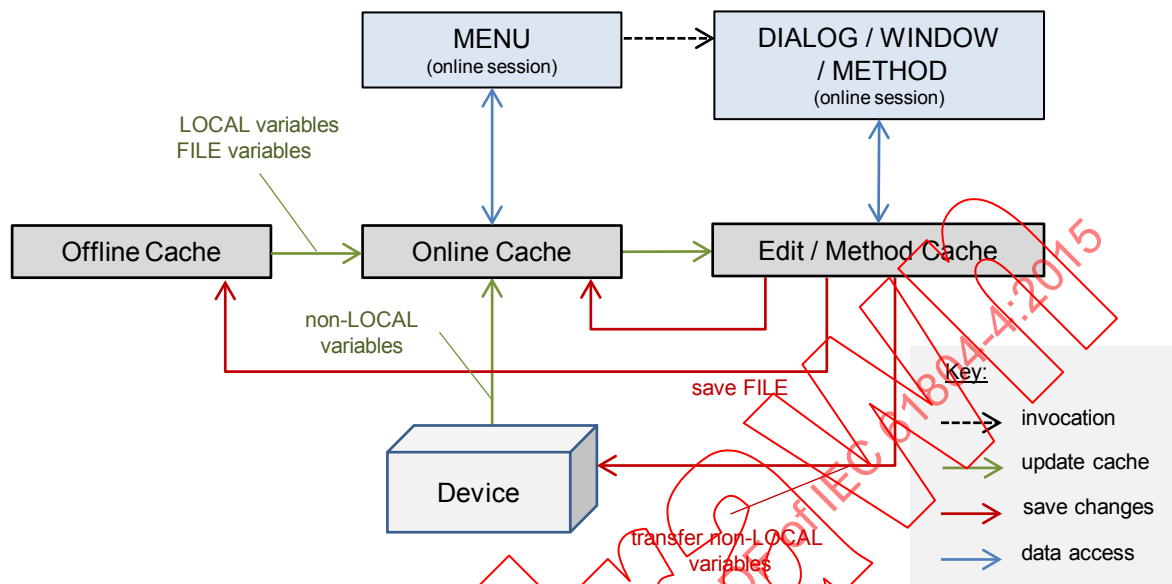


Figure 82 – Data caching for an online session

After transferring to the device, the transferred variables and the affected VARIABLES of UNIT and REFRESH relations shall be reread if requested.

When the dialog or window is confirmed, the EDD application shall validate the values of the data items. Only if the values are inside of the MIN/MAX_VALUE ranges the EDD application shall allow the data to be transferred to the device. When a METHOD finishes without abort and the required Builtins are invoked, the changes shall be saved according to 8.1.4. A METHOD can communicate with the device at anytime by using communication Builtins. Changes on the values of VARIABLES referenced in FILES shall be saved into the offline cache to be stored persistently. Successfully transferred value to the device shall be stored in the online cache.

8.1.2 General rules

All caches shall contain unscaled values even if the VARIABLES have a SCALING_FACTOR attribute. The variable values shall only be scaled to be displayed and unscaled after editing.

Any conditionals shall be evaluated using the current cache.

FOUNDATION fieldbus caches shall support multiple block instances.

8.1.3 Data caching for dialogs and windows

When the user invokes a dialog or a window (MENU with STYLE DIALOG or STYLE WINDOW), the EDD application shall use an edit cache with values from the online or offline cache.

Sub dialogs or windows may share the edit cache of the dialog or window they are invoked from. Changes within a sub dialog or window are applied directly to the invoking dialog or window and cannot be discarded.

Figure 83 shows the data caching for sub dialogs or windows using a shared edit cache.

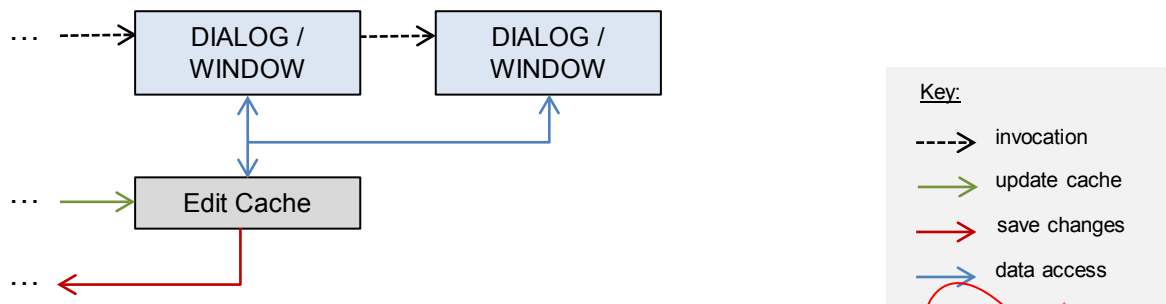


Figure 83 – Sub dialogs or windows using a shared edit cache

If the EDD application uses separate edit caches for sub dialogs and windows, the changes are applied to the invoking dialog or window if the sub dialog or window is confirmed, otherwise the changes are discarded.

Figure 84 shows the data caching for sub dialogs or windows using separate edit caches.

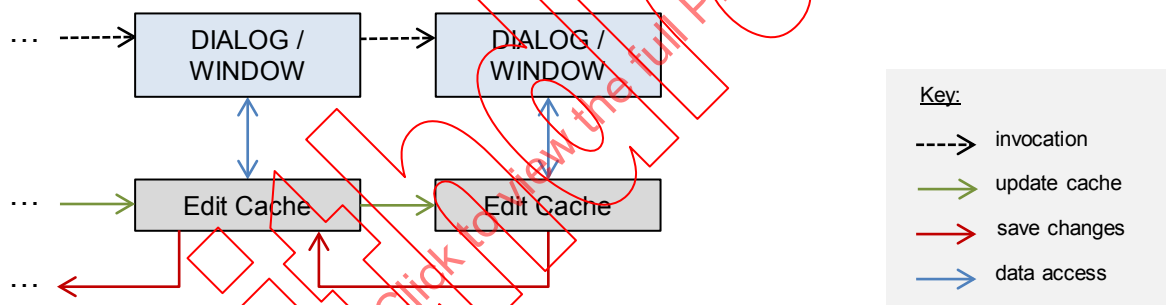


Figure 84 – Sub dialogs or windows using separate edit caches

8.1.4 Data caching for METHODS

When the user invokes a METHOD, the EDD application shall use a separate method cache based on the cache of the MENU the METHOD has been invoked from. METHOD local variables are not stored in the method cache.

METHODs called from a METHOD shall share the method cache of the calling METHOD. If a nested METHOD aborts, the whole calling hierarchy shall be aborted and all changes shall be discarded. The saving of changes within a METHOD is described in Table 12. Figure 85 shows the data caching for nested METHODS.

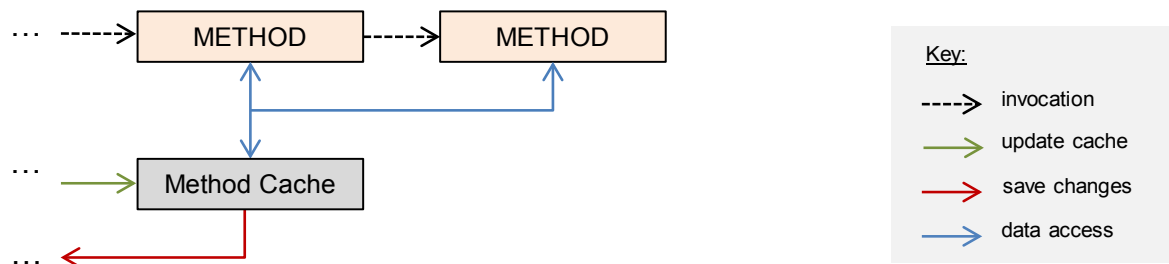


Figure 85 – Data caching for nested METHODS

When a METHOD is invoked from a dialog or window, the EDD application shall use a method cache based on the cache of the MENU the METHOD has been invoked from.

Figure 86 shows the data caching for METHODS invoked from a dialog.

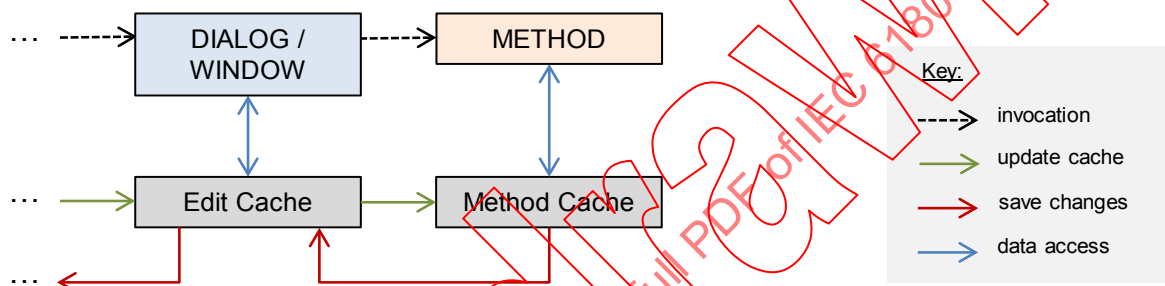


Figure 86 – Data caching for a METHOD invoked within a dialog

When a dialog is invoked from a METHOD by calling the Builtin MenuDisplay, the EDD application may use an edit cache with values from the method cache of the calling METHOD.

Figure 87 shows the data caching for a dialog invoked from a METHOD with a separate edit cache.

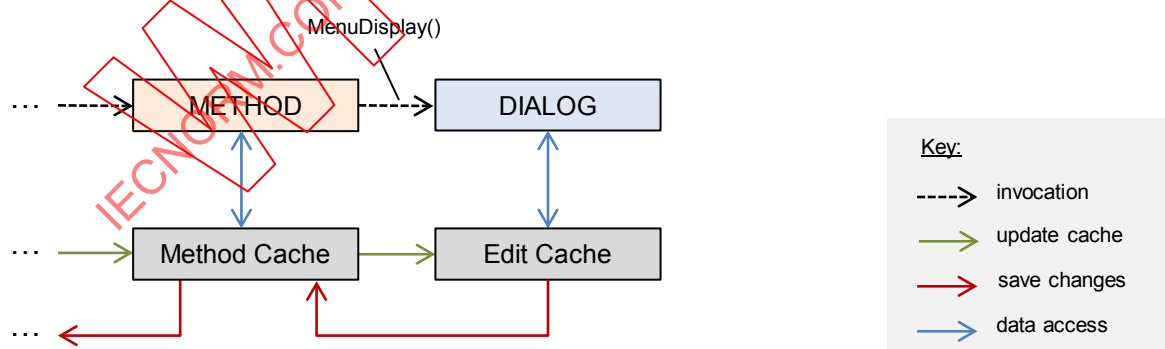


Figure 87 – Data caching for a METHOD invoking a dialog using an edit cache

Figure 88 shows the data caching for a dialog invoked from a METHOD sharing the method cache of the calling method as edit cache.

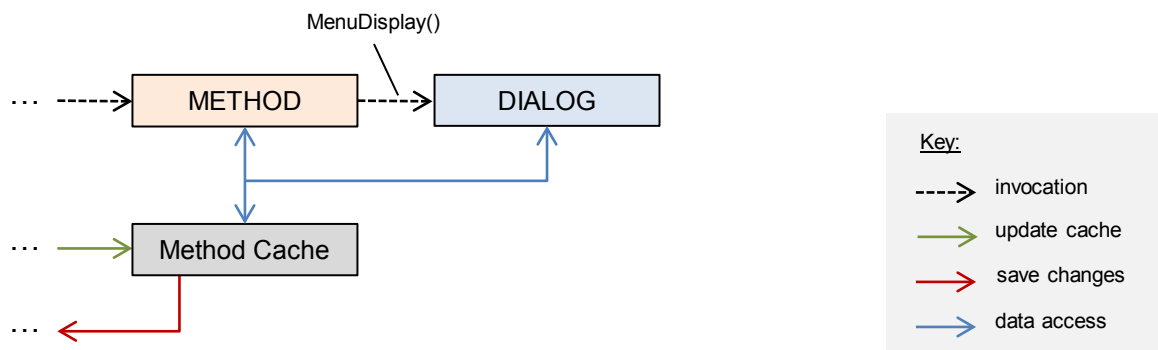


Figure 88 – Data caching for a METHOD invoking a dialog

HART and FOUNDATION fieldbus do not support invoking a METHOD from a dialog, which is invoked from a METHOD by calling the Builtin MenuDisplay.

Modifications of LIST structures by calling the Bultins ListInsert and ListDeleteElementAt shall remain even if the method aborts.

In METHODS invoked by the user, the Builtin save_values or save_on_exit controls saving the changes of the method cache. Table 12 shows the usage of the Bultins (see IEC 61804-5). Methods can also transfer values from and to the device by calling communication Bultins.

Table 12 – Bultins for method cache controlling

Builtin	HART	FOUNDATION fieldbus	PROFIBUS, PROFINET	Cache handling
save_values	Required	—	Optional	All changes shall be saved immediately.
save_on_exit	—	Required	Default behavior	All changes shall be saved when the method finishes correctly.
discard_on_exit	Optional	Optional	Optional	Discards all values at the end of the method.
send_on_exit	—	Optional	Optional	All changes shall be saved and transferred to the device when the method finishes correctly.
send_all_values	—	Optional	—	All changes shall be immediately saved and transferred to the device.
send_value	—	Optional	—	The value shall be immediately saved and transferred to the device.
Key: Required: calling the Builtin is required to save the changes Optional: calling the Builtin is not required to save, send or discard the changes Default behavior: calling the Builtin is not necessary because behavior of the Builtin is equal to the behavior without calling the Builtin —: calling of the Builtin has no effect or is not supported				

Actions should use the cache they are invoked from. In METHODS of VARIABLE actions, action Bultins are used to get or set the value of the variable for which it is invoked. It is not necessary to call any other cache handling Builtin to save the changes; the set_ (or put_) Bultins will save this computed value. If any variables other than the action variable are modified, cache controlling Bultins will need to be used to designate whether these variable values should be saved, sent to the device or discarded.

8.2 UI aspects of editing sessions

User interfaces described in an EDD imply a common editing behavior. The following rules shall be implemented by the EDD application.

- a) VARIABLES shall be displayed with a variable state that indicates that the value is out of range or in case of ENUMERATED and BIT_ENUMERATED that the value has no related enumerator. Some checks may not be done in the EDD and shall be done in the device. This leads to a communication response_code not equal to SUCCESS.
- b) VARIABLES shall be marked, if the value is changed by the user or indirectly by METHODS.
- c) The EDD application may show additional indications. After transfer to the device or save to the offline data base, a variable is no longer marked.
- d) The user interface shall be updated depending on the changes of variable that are used in conditional expressions. Examples are:
 - 1) Conditions in ENUMERATED or BIT_ENUMERATED VARIABLES and conditions in MIN_VALUES and MAX_VALUES of numeric VARIABLES shall be considered.
 - 2) The layout may change depending on attributes like VISIBILITY, VALIDITY and MENU ITEMS.
 - 3) The HANDLING attribute shall be considered.
 - 4) Conditions in COLLECTION MEMBERS, the changing number of items in LIST, etc. shall be considered.
 - 5) Conditions in the PATH attribute of IMAGES shall be considered
- e) The EDD application may allow that the user enters invalid numeric values that are outside the MIN/MAX_VALUE ranges but are inside the range of the data type to support complex dependences. The EDD application shall require explicit acknowledgement from the user when entering one or more values outside the MIN/MAX_VALUE ranges. While editing ENUMERATED and BIT_ENUMERATED VARIABLES the user shall not be able to enter invalid values.
- f) The EDD application shall activate the same instance of a WINDOW that uses already the same data cache. The EDD application shall invoke a new instance of this WINDOW, if it is activated by an EDD instance uses a different data cache.
- g) All sub windows of a DIALOG shall be handled like a DIALOG.
- h) A MENU of STYLE DIALOG is modal, that means, no interactions outside of the DIALOG shall be possible.
- i) If in an online session a cause variable of a UNIT or REFRESH relation is changed, the EDD application should mark the affected variables.

8.3 User roles

EDDL supports a role concept that allows EDD developers to mark variable and methods as EDD content that shall not be provided to all users. This can be defined in EDD with CLASS attribute of VARIABLES or METHODS and the class SPECIALIST. This can be used to prevent users to make changes or invoke functions by accident.

To ensure the maintenance and specialist variables and methods meet the needs of the customer, the EDD application may allow the specialist to override the CLASS attribute in order to tailor the list of variables.

9 Offline and online configuration

9.1 Overview

The life cycle of a plant contains different phases. In the design and engineering phase the engineer wants to specify the required device behavior without having the possibility to access the devices. The offline configuration supports this use case.

Device parameters can be differentiated into application-specific parameters and device-specific parameters. To use a new device at a plant location, the application-specific

parameter should be downloaded into the device. When a device is replaced, the application-specific parameters from the old device should be downloaded into the new device. However, the device-specific parameters are never downloaded because they are specific to the physical device.

9.2 Offline dataset

The offline dataset contains data item values. Any data item except VARIABLES of CLASS TEMPORARY referenced by the offline_root_menu, other offline MENUs and offline METHODS shall be stored in the offline dataset.

9.3 Offline configuration

The offline configuration is the sum of all activities that change the offline dataset. The offline configuration is specified by e.g. offline_root_menu, upload/download, offline MENUs and METHODS, BLOCK_A PARAMETERS, and COMPONENT features like CHECK_CONFIGURATION, SCAN and DETECT.

Devices may not or only partly support offline configuration. In this case the offline_root_menu and offline menus and methods may not exist or not contain all necessary data items to fulfill a complete configuration.

Any conditionals evaluated during the offline configuration shall be evaluated using the offline dataset.

The offline configuration shall not require online access that is not explicitly chosen by the user e.g. by a data item action. Such online access user interfaces in offline configuration can exist through METHODS which are using communication Builtins and MENUs or METHODS with ACCESS ONLINE.

9.4 Online dataset

The online dataset contains variable values read by the EDD application from the device. LOCAL VARIABLES used in the online configuration shall be also in the online dataset.

9.5 Online configuration

The online configuration is specified by:

- device_root_menu
- diagnostics_root_menu
- maintenance_root_menu
- process_variables_root_menu
- root_menu (Handheld)
- Menu_Top* (Handheld)
- hh_* (Handheld)
- BLOCK_A PARAMETERS and LOCAL_PARAMETERS

Any conditionals evaluated during the online configuration shall be evaluated using the online dataset.

9.6 Upload and download

9.6.1 Overview

An EDD application shall support the upload procedures to transfer data from the device into the offline dataset and the download procedures to transfer data from the offline dataset to the device.

The ordering in the offline configuration menu (`offline_root_menu`) is user-aspect-orientated in contrast to the upload or download menus that are ordered from a communications point of view.

Upload and download menus should contain all of the application-specific parameters for the device. For special handling of errors, timing of the commands, etc., the upload and download menus may contain also METHODS.

METHODs invoked while uploading or downloading should not require any user interactions.

Figure 89 shows the data flow and the related caches for download to the device.

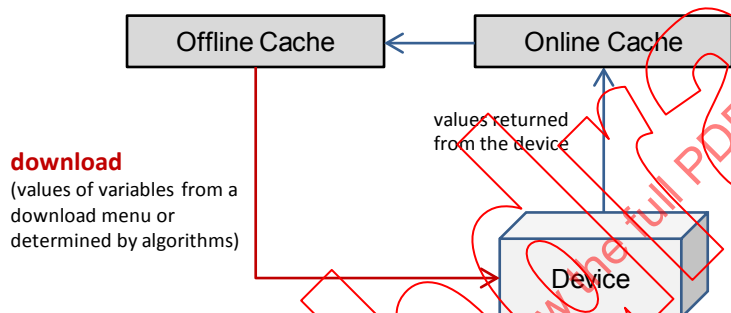


Figure 89 – Data flow for download to the device

Figure 90 shows the data flow and the related caches for upload from the device.

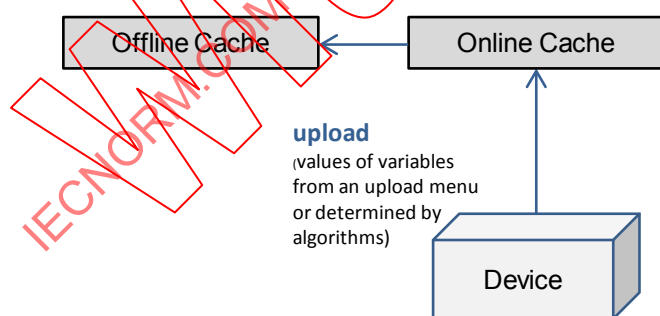


Figure 90 – Data flow for upload from the device

The upload and download menus should not contain menu item qualifiers, e.g., HIDDEN, READ_ONLY, etc. An EDD application shall ignore any menu item qualifiers that appear within upload and download menus.

Some devices support special parameters that can only be written to the device when the device is in a particular operating mode. The MENU PRE_WRITE_ACTIONS may be used to place the device into the proper operating mode, and the MENU POST_WRITE_ACTIONS may be used to return the device to its original mode.

9.6.2 Error recovery

If an error response code is returned by the device during a download/upload process, the error should be logged and the process should be aborted. If a warning response code is returned by the device a warning should be logged and the process should continue. The errors and warning should be logged in a manner consistent with the LOG_MESSAGE() Builtin function. The EDD developer should put in place appropriate abort methods to recover from errors and restore the device to a well known state.

If a download is canceled, the device may not be in a consistent, well defined state.

9.6.3 Upload procedure

9.6.3.1 Overview

An EDD may contain an upload menu by using one of the identifiers in Table 13 in order to determine which data items should be read from the device.

The EDD application shall support the identifiers in Table 13. If the upload_from_device_root_menu exists, it shall be used, otherwise if the download_variables menu exists, it shall be used, otherwise the upload without menu procedure in 9.6.3.4 shall be used.

Table 13 – List of defined upload menu identifiers

Menu identifiers
upload_from_device_root_menu
download_variables

9.6.3.2 General rules

The following are the general rules for the upload procedure.

- Any conditional attributes (e.g. the ITEMS attribute of a MENU) should be evaluated using the data read from the device.
- If a COMMAND reads multiple variables, the read order can be affected.
- If the VARIABLE has already been read it should not be read again.
- If during the upload, errors or warnings occur, the EDD application shall provide a mechanism to inform the user. RESPONSE_CODES type defines how to interpret return information from the device. In IEC 61804-3 errors, warnings and success are defined.
- If an error is returned from the device, the upload procedure should be canceled.
- A VARIABLE with VALIDITY FALSE should not be read.
- The PRE_READ_ACTIONS of the VARIABLES shall be executed immediately before and the POST_READ_ACTIONS after reading the VARIABLE from the device.
- If one of the variable PRE_READ_ACTIONS or POST_READ_ACTIONS methods aborts, the upload procedure should continue.
- If one of the menu PRE_READ_ACTIONS methods aborts, the upload procedure of this menu should be canceled.
- If one of the menu POST_READ_ACTIONS methods aborts, the upload procedure of the menu should continue.
- If actions abort, the user shall be informed.

9.6.3.3 Upload with upload menu

If an upload menu is defined, the upload shall be proceeded in following steps:

- 1) if the VALIDITY of a menu is evaluated to FALSE, the following steps shall be skipped;
- 2) the PRE_READ_ACTIONS methods of the upload menu shall be executed. If any PRE_READ_ACTIONS method aborts, the upload procedure shall be aborted;
- 3) the order of the items to be read shall be determined according to the upload menu. Data items should be read from the device in the order in which they appear in the upload menu. If the item is a MENU, the sub-menu should be processed in the same way as the upload menu.

If an EDD defines the MENU upload_to_device_root_menu, the upload procedure shall transfer all data items necessary to configure a device with a download e.g. after a device replacement or device reset.

9.6.3.4 Upload without upload menu

If no upload menu is defined, the upload shall be proceeded in following steps:

- 1) The EDD application shall generate a list of data items to be read from the device containing all the data items referenced in the read COMMANDs, i.e. COMMANDs with "OPERATION READ", except VARIABLES of CLASS LOCAL.

For BLOCK_A devices the EDD application shall read items in the PARAMETERS attribute.

- 2) The order of the items to be read shall be determined according to VARIABLES needed by conditionals, UNIT and REFRESH relations. The cause-parameters of UNIT and REFRESH relations should be read before the effected-parameters. The ordering of items in unit and refresh relations cause or effect lists are not used for ordering the communication.

Any data item not ordered by the rules in 9.6.3 can be read in any order. Devices shall not assume a specific order.

9.6.3.5 Using online dataset for upload

While the EDD application is connected to the device and in online configuration, data items are cached in the online dataset. When the EDD application switches from online to offline configuration any data required from the device for offline configuration shall be used from the cached online dataset. Any required online data that is not cached shall be uploaded from the device using the upload procedure in 9.6.3.

9.6.4 Download procedure

9.6.4.1 Overview

An EDD may contain a download menu by using one of the identifiers in Table 14 in order to determine which data items should be written to the device.

The EDD application shall support the identifiers in Table 14. If the download_to_device_root_menu exists, it shall be used, otherwise if the upload_variables menu exists, it shall be used, otherwise the download without menu procedure in 9.6.4.4 shall be used.

If an EDD defines the MENU download_to_device_root_menu, this menu shall define the transfer all data items necessary to configure the device.

Table 14 – List of defined download menu identifiers

Menu identifiers
download_to_device_root_menu
upload_variables

9.6.4.2 General rules

The following are the general rules for the download procedure.

- Any conditional attributes (e.g. the ITEMS attribute of a MENU) should be evaluated using data from the offline dataset.
- The download should have no effect on the offline dataset. However, if the device stores a value different than the one in the offline dataset, the difference should be detected. In this case the user should have the opportunity to decide whether the value from the device should be carried over to the offline dataset (see Figure 89).
- If a COMMAND writes multiple variables, the write order may be affected.
- If the VARIABLE has already been written it should not be written again.
- If during the download, errors or warnings occur, the EDD application shall provide a mechanism to inform the user. The RESPONSE_CODES type defines how to interpret return information from the device. In IEC 61804-3, errors, warnings and success are defined.
- If an error is returned by the device, the download procedure should be aborted.
- A VARIABLE with VALIDITY FALSE should not be written.
- The PRE_WRITE_ACTIONS of the VARIABLES shall be executed immediately before and the POST_WRITE_ACTIONS after writing the VARIABLE to the device.
- If one of the variable PRE_WRITE_ACTIONS or POST_WRITE_ACTIONS methods aborts, the download procedure should continue.
- If one of the menu PRE_WRITE_ACTIONS methods aborts, the download procedure of this menu should be canceled.
- If one of the menu POST_WRITE_ACTIONS methods aborts, the download procedure of the menu should continue.

9.6.4.3 Download with download menu

If a download menu is defined, the download shall be proceeded in following steps:

- 1) The EDD application shall validate that every variable to be downloaded has a valid value, as specified by the EDD. If an invalid value has been found, the EDD application shall not start the download unless the user has allowed it.
- 2) The PRE_WRITE_ACTIONS methods of the download menu shall be executed. If any PRE_WRITE_ACTIONS method aborts, the download procedure shall be aborted.
- 3) The order of the items to be written shall be determined acc. the download menu. VARIABLES, LISTS, RECORDS, or value arrays should be written to the device in the order in which they appear in the download menu. If the item is a MENU, the sub-menu should be processed in the same way as the download menu.
- 4) When using the upload_variables menu for HART, all remaining writable data items shall also be downloaded to the device.

9.6.4.4 Download without download menu

If no download menu is defined, the download shall be proceeded in following steps:

- 1) The EDD application shall generate a list of data items to be written to the device containing all data items referenced in the write COMMANDs, i.e. COMMANDs with "OPERATION WRITE", excepting VARIABLES of CLASS LOCAL or DYNAMIC. The cause-parameters of UNIT and REFRESH relations should be written before the effected-parameters.
- 2) The EDD application shall validate that every variable to be downloaded has a valid value, as specified by the EDD. If an invalid value has been found, the EDD application shall not start the download unless the user has allowed it.

- 3) The order of the items to be written shall be determined according to UNIT and REFRESH relations. The cause-parameters of UNIT and REFRESH relations should be written before the effected-parameters.

Any data item not ordered by the rules in 9.6.4 can be written in any order. Devices shall not assume a specific order. The ordering of items in unit and refresh relations cause or effect lists are not used for ordering the communication.

The following rules apply to FOUNDATION fieldbus devices.

- The EDD application should enable the “deferral of inter parameter write checks” feature in the device prior to the download, if supported.
- The EDD application shall calculate the appropriate download target mode according to each parameter's WRITE_MODE attribute of the offline dataset. If this attribute is not specified, then MODE_OUT_OF_SERVICE shall be used. The EDD application may allow the user to define a different download target mode. The download target mode parameter for each BLOCK_A shall be written prior to the download. The final target mode is specified in the offline data set and shall be the last parameter transferred to a BLOCK_A.
- Parameters listed in a no_download* (see Table B.1) COLLECTION as specified for each BLOCK_A COLLECTION_ITEMS attribute shall be not part of a download.

10 EDDL communication description

10.1 COMMAND

10.1.1 General

A COMMAND specifies the data being transferred from and to a device. The data to be written to the device is specified in the REQUEST attribute. The data to be read from the device is specified in the REPLY attribute.

COMMANDs can be used for different communication profiles (CPs), see IEC 61784-1 and IEC 61784-2. Some attributes are CP specific.

A COMMAND with any invalid data items (VALIDITY FALSE) in the REQUEST or where all data items are invalid in the REPLY shall not be used by the EDD application. Only valid data items replied to shall be updated, invalids shall be ignored. The EDD application shall abort a method that forces a COMMAND with that condition.

10.1.2 OPERATION

10.1.2.1 General

This attribute specifies the purpose of the command.

10.1.2.2 OPERATION READ

OPERATION READ specifies that the primary purpose of this COMMAND is to read data items from the device. But, depending on the protocol or device specific addressing mechanisms a READ command can specify data to be transferred to the device e.g. INFO, INDEX variables before data is received from the device. With this data, the device can address the data items that should be transferred to the EDD application.

10.1.2.3 OPERATION WRITE

OPERATION WRITE specifies that the COMMAND writes data items to the device. A WRITE command can specify data in the REPLY attribute that is transferred from the device after writing. If the REPLY attribute contains data items that are written to the device with this command, data modifications made by the device can be detected in the EDD application.

10.1.2.4 OPERATION COMMAND

OPERATION COMMAND specifies that the COMMAND purpose is to trigger a function. REQUEST specifies the input parameters. REPLY specifies the output parameter. The EDD application shall not use such COMMANDs for data transfer. Such COMMANDs may only be used by using communication Builtins in METHODS.

10.1.2.5 PROFIBUS and PROFINET communication mapping

Following Table 15 defines for PROFIBUS and PROFINET the communication mapping to the communication services.

Table 15 – PROFIBUS and PROFINET communication mapping

REQUEST items	REPLY items	command
no	no	EDD application shall use a write service
yes	no	EDD application shall use a write service
no	yes	EDD application shall use a read service
yes	yes	Is not allowed. The EDD application shall generate an error and not perform any communication.

10.1.3 TRANSACTION

10.1.3.1 General

A TRANSACTION specifies the data to be transferred. A COMMAND can contain one or more TRANSACTIONS.

If the COMMAND has multiple TRANSACTIONS, each is identified by an additional COMMAND specific unique identifier. All transactions have the same COMMAND addressing attributes and the differentiation shall be defined by the REQUEST data.

10.1.3.2 REPLY and REQUEST

REQUEST specifies the data frame that is sent to the device. REPLY specifies the data frame that is received from the device.

REPLY and REQUEST are lists of data items. Data items may be constants or references to variables, lists, arrays, or collections. The referenced EDD items of REFERENCE_ARRAYs or COLLECTIONs shall be evaluated to data items such as VARIABLEs or VALUE_ARRAYs.

Constants in the REQUEST data item list are used if the data is not configurable or as an additional identification information (addressing) for the device.

Constants in the REPLY data item list means that data from the device will not be used in that COMMAND.

10.1.3.3 Data item masks

Item masks are only needed when VARIABLEs or constants do not begin and end on byte boundaries. An item without an item mask starts at bit 0. Item masks may be specified for VARIABLEs and constants. The bits with the value '1' in the item mask defines the mapping of the data item to the data frame. The bits with the value '1' shall be consecutive. Gaps in the item mask are not allowed. If the value of bit 0 of the item-mask is '1', the following item is mapped to bit 0 of the next byte in the data frame. Bits not contained in the item mask are set to 0. Overlapping between item masks is not allowed, it is only allowed if bit 0 of the previous mask is set.

If using HART no gaps between item masks shall be used.

Multi byte item masks define a mapping to a larger range in the data frame. The number of bits with the value '1' in the item mask shall be less than or equal to the number of bits of the data item.

10.1.3.4 Example of usage of a single item mask

If an item mask is specified for a VARIABLE or constant and the previous and next item do not have an item mask, then the first bit set in the mask defines the start position and the last bit set the end position.

```

COMMAND cmd_abc
{
    OPERATION WRITE; // or READ
    ...              // protocol specific attributes, e.g. SLOT and INDEX
    TRANSACTION
    {
    REQUEST          // or REPLY in case of READ
    {
        a,
        b <0x3c>,    // 00111100b
        c
    }
    }
}

```

Figure 91 – Example of a single item mask

In Figure 91 a, b and c are INTEGER(1) VARIABLES. In this case bit 2 to bit 5 of VARIABLE b are mapped to bit 0 to bit 3 behind VARIABLE a. VARIABLE c starts on bit 0 of the next byte in the data frame (see Figure 92).

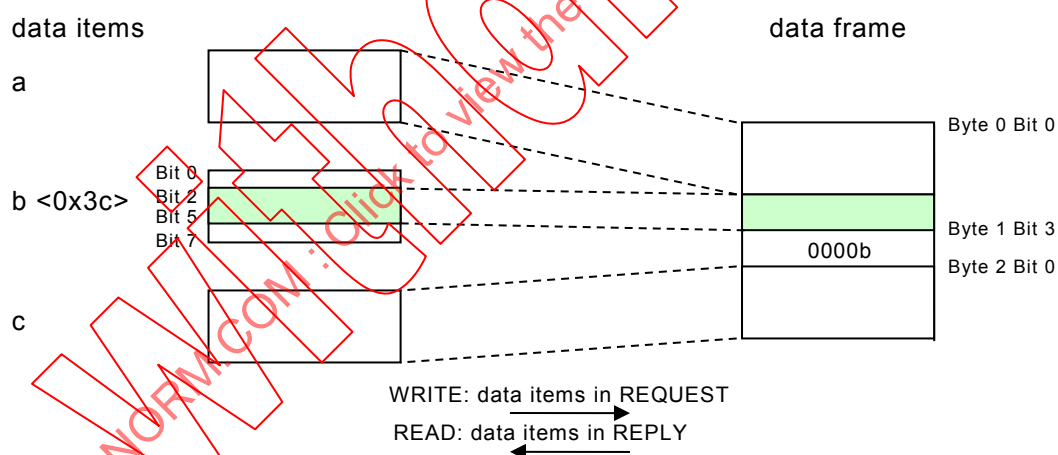


Figure 92 – Mapping example with a single item mask

The EDD example Figure 93 and Figure 94 show the usage of multiple item masks.

```

a,
b    <0xFFF0>,    // 1111111111110000b
c    <0x000C>,    // 0000000000001100b
0    <0x0002>,    // 0000000000000010b, required only in HART, because of item mask gaps
d    <0x0001>,    // 0000000000000001b
e,

```

Figure 93 – Multiple item masks

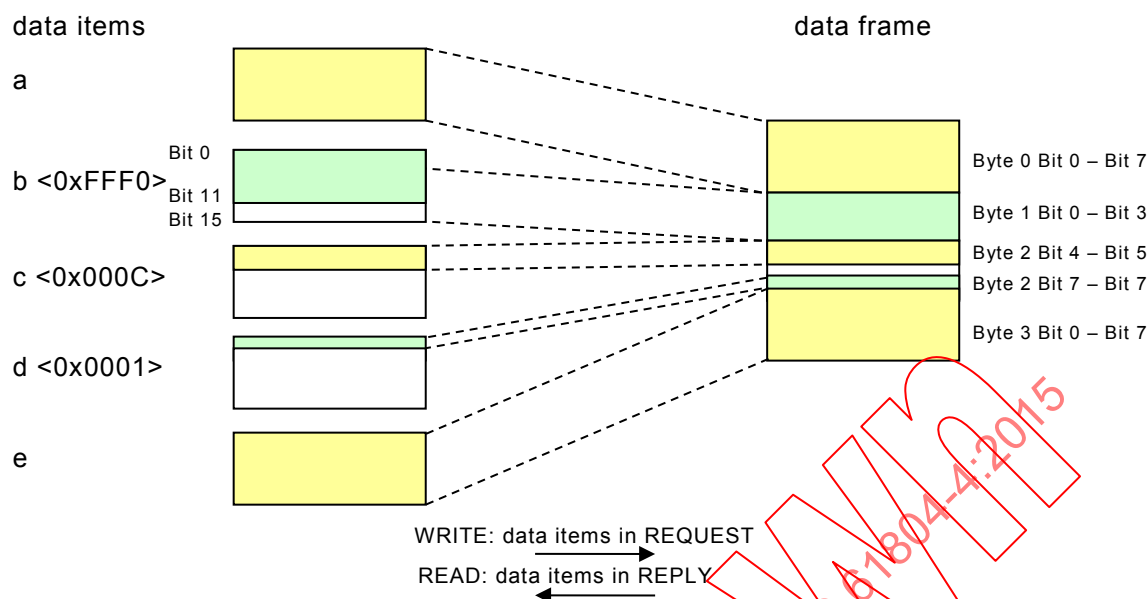


Figure 94 – Mapping example with a multiple item mask

10.1.3.5 Data item qualifiers

A VARIABLE appearing in a request or reply may be qualified with the keywords INDEX and INFO.

A VARIABLE qualified with INDEX specifies that the VARIABLE is used in the request or reply as array index.

A VARIABLE qualified with INFO specifies the variable is not actually stored in the device, the VARIABLE is only communicated for informational purposes.

A VARIABLE may be qualified with both INDEX and INFO; in this case, the VARIABLE is used as array index and is not actually stored in the device.

Figure 95 shows an EDD example demonstrating the use of an INFO qualified VARIABLE.

```
TRANSACTION
{
  REQUEST
  {
    units (INFO), x, y
  }
  REPLY
  {
    units, x, y
  }
}
```

Figure 95 – INFO qualifier

The VARIABLES x and y are written to the device in the unit which is specified by 'units'. The VARIABLES x and y are stored in the device (presumably in a fixed unit), but 'units' is not. This allows a VARIABLE to be communicated in a unit different from its current unit (the unit in which the variable is displayed).

Figure 96 shows an EDD example demonstrating the use of an INDEX qualified VARIABLE.

```
TRANSACTION
{
  REQUEST
  {
    code (INDEX), table[code]
  }
  REPLY
  {
    code, table[code]
  }
}
```

Figure 96 – INDEX qualifier

The data item in the array named table specified by a code is written to the device. The device stores the value of the code and echoes its value. The data item sent along with the code is also stored and the value stored in the device is returned.

Figure 97 shows an example demonstrating the use of local index VARIABLES.

```
TRANSACTION
{
  REQUEST
  {
    code (INFO, INDEX), table[code]
  }
  REPLY
  {
    code, table[code]
  }
}
```

Figure 97 – INFO and INDEX qualifier

The VARIABLE 'code' with the INFO and INDEX qualifier is sent to the device, but not stored in the device. The VARIABLE is only used as index for the array 'table'.

10.1.3.6 RESPONSE_CODE

Response codes specify the values the device returns as the response code. Response codes appearing within a TRANSACTION apply only to that TRANSACTION, while response codes appearing outside of any TRANSACTION apply to all TRANSACTIONS. If the same response code is specified both inside and outside of a TRANSACTION, the response code specified within the TRANSACTION takes precedence, but only for that TRANSACTION.

Each (value, type, description, help) quadruple specifies one response code.

- The first component, value, is an integer constant that specifies the response code value.
- The second component, type, is the response code type.
- The third component, description, is a string that specifies the text that shall be displayed when the response code is returned by the device.
- The last component, help, is a string which provides a moderately extensive description of the response code that may be displayed.

10.1.4 Command addressing

10.1.4.1 HART Command addressing

The NUMBER specifies the HART command number that the device use to identify the command and the data.

10.1.4.2 PROFIBUS DP addressing

SLOT and INDEX specify the PROFIBUS dataset that is transferred. In a modular device the modules are typically addressed with the SLOT and the data of a module is addressed with the INDEX.

10.1.4.3 PI Profile for Process Control Devices Addressing

The absolute addresses of BLOCKs of a PI Profile for Process Control Devices device are stored in the device management directory. The EDD application has to fetch the directory and to resolve the base address of the BLOCK (BLOCK_B). The BLOCK attribute of a COMMAND is a reference to a BLOCK_B item.

A BLOCK_B EDDL item is described with a TYPE and a NUMBER attribute. The TYPE attribute defines the types of a block (PHYSICAL, TRANSDUCER or FUNCTION). The NUMBER attribute defines the number of a block of a block type in the device management directory. NUMBER starts for all block types with 1.

The command attribute INDEX is a relative index to the first dataset of the block. To address a dataset in the device, the absolute starting slot and starting index from the device management directory are used and the relative INDEX of the COMMAND is added.

10.1.4.4 PROFINET addressing

API, SLOT, SUB_SLOT and INDEX are used to address datasets in the device in an absolute way.

10.1.4.5 Addressing for other protocols

HEADER is a string attribute that allows to contain protocol specific information for a COMMAND. The HEADER attribute is not used for PROFIBUS, PROFINET, FOUNDATION fieldbus or HART devices. The concrete contents of the string is out of the scope of this standard.

10.2 Parsing data received from the device

10.2.1 General

The EDD application shall transfer data received from the device into the requested data items. If the device returns exactly the same amount of data as requested and doesn't return an error or warning (see also RESPONSE_CODE), the EDD application shall not log an error.

The EDD application shall not change data items which are not received from the device. EDDL application shall not invoke POST_READ_ACTIONS for data items not received.

10.2.2 Parsing complex data items

When a LIST, VALUE_ARRAY, REFERENCE_ARRAY, COLLECTION and RECORD is referenced by itself in a communication (e.g. COMMAND), the EDD application shall fill up data from the device into referenced data item with the following rules:

- a) LIST, VALUE_ARRAY and REFERENCE_ARRAY
shall be filled up starting at the lowest index value
- b) COLLECTION and RECORD
shall be filled up from the first MEMBER

10.2.3 FOUNDATION Fieldbus

The EDD application shall log an error if the device returns less data than specified by RECORDs, VALUE_ARRAYs, PARAMETER_LISTs or VARIABLES.

If the device returns more data than a `PARAMETER_LIST` specifies, the EDD application shall ignore additional data without logging an error or warning. For all other data items the EDD application shall log an error if more data is received.

10.2.4 HART

The EDD application shall log a fatal error if the device returns less data than specified by the `COMMAND`.

If the device returns more data than the `COMMAND` specifies, the EDD application shall ignore additional data without logging an error or warning.

EDDL application shall not invoke `POST_WRITE_ACTIONS` for data items not received after a write `COMMAND`.

10.2.5 PROFIBUS and PROFINET

The EDD application shall log an error if the device returns less data than specified by the `COMMAND REPLY` unless the last data item is a variable of any string data type. The EDD application shall transfer all received data into this variable.

If the device returns more data than the `COMMAND REQUEST` specifies, the EDD application shall ignore additional data without logging an error or warning unless the last data item is a `LIST`. If the last data item is a `LIST`, the EDD application shall use or create `LIST` elements until all data from the device are consumed.

10.3 FOUNDATION Fieldbus communication model

FOUNDATION fieldbus devices are block object oriented devices according to IEC 61804-2. Each block is composed of a characteristics and one or more block parameters. In EDDL, each block type is defined using `BLOCK_A` construct. The `BLOCK_A` attribute `CHARACTERISTICS` maps to the block characteristics found in the device. Block parameters can be `VARIABLES`, `RECORDs` and `VALUE_ARRAYs`.

One or more blocks are represented in an object dictionary. The object dictionary is accessed by an index number using read and write services. The object dictionary provides a block directory to identify the number and location of all blocks within the object dictionary.

All data items accessible by an EDD application are accessed in the context of a block. The EDD does not provide any information about the object dictionary index of blocks. Instead, the block characteristics provides a reference to the EDD item using a unique symbol id.

When an EDD is created, unique symbol ids are generated for each EDD item. This symbol id is also encoded in the block characteristics of each block in the device.

By accessing the characteristics of each block to find the associated symbol id, the EDD application associates an EDD `BLOCK_A` declaration with the block in the device. All `BLOCK_A PARAMETERS` are listed consecutively after the block `CHARACTERISTICS`.

By accessing the object dictionary description and block directory, the EDD application can calculate the absolute index to access each `BLOCK_A CHARACTERISTICS` and `PARAMETERS`.

A device may have several instances of blocks that will map to a single `BLOCK_A` declaration in the EDD.

Figure 98 illustrates 2 instances of `BLOCK b1` and 1 instance of `BLOCK b2` in an object dictionary. The Figure 99 EDDL fragment can be used to describe this example.

Index	Description	
0	Object Dictionary Object Description	
...		
n	Block Directory	
...		
p	BLOCK b1 CHARACTERISTICS cb1	BLOCK b1 [0]
p+1	BLOCK b1 Parameter P1 (va)	
p+2	BLOCK b1 Parameter P2 (vb)	
...	...	
q	BLOCK b1 CHARACTERISTICS cb1	BLOCK b1 [1]
q+1	BLOCK b1 Parameter P1 (va)	
q+2	BLOCK b1 Parameter P2 (vb)	
...	...	
r	BLOCK b2 CHARACTERISTICS cb2	BLOCK b2
r+1	BLOCK b2 Parameter P1 (va)	
r+2	BLOCK b2 Parameter P2 (vc)	
...	...	

Figure 98 – Example device with 2 unique BLOCK_A definitions

```

BLOCK b1
{
    CHARACTERISTICS cb1;
    PARAMETERS
    {
        P1, va;
        P2, vb;
        /* ... */
    }
}

BLOCK b2
{
    CHARACTERISTICS cb2;
    PARAMETERS
    {
        P1, va;
        P2, vc;
        /* ... */
    }
}

VARIABLE va { /* ... */ }
VARIABLE vb { /* ... */ }
VARIABLE vc { /* ... */ }

RECORD cb1
{
    MEMBERS
    {
        /* ... */
        DD_ITEM, __dd_item; /* symbol id of block b1 */
        /* ... */
    }
}

RECORD cb2 { /* ... */ }

```

Figure 99 – Example EDD for a device with 2 unique BLOCK_A definitions

Each block also references 4 block views. Block views provides read and write access to several block parameters at a single object dictionary index. Block views provide an efficient means to access multiple parameters from a device in a single communication transaction.

Each view is described in EDDL using the VARIABLE_LIST construct. Each VARIABLE_LIST is associated with a BLOCK_A using the PARAMETER_LIST attribute. The BLOCK CHARACTERISTICS provides mapping to the absolute index of the view. Each view is consecutively listed in the object dictionary

Figure 100 illustrates an example of VARIABLE_LIST vl1 for BLOCK b1. The Figure 101 EDDL fragment can be used to describe this example.

Index	Description
0	Object Dictionary Object Description
...	
n	Block Directory
...	
p	BLOCK b1 CHARACTERISTICS cb1
p+1	BLOCK b1 Parameter P1
p+2	BLOCK b1 Parameter P2
p+3	BLOCK b1 Parameter P3
p+4	BLOCK b1 Parameter P4
v	BLOCK b1 VIEW_1 (vl1)
v+1	BLOCK b1 VIEW_2 (vl2)
v+2	...
v+3	...

Figure 100 – BLOCK_A example with PARAMETER_LISTS

```

BLOCK b1
{
    CHARACTERISTICS cb1;
    PARAMETERS
    {
        P1, va;
        P2, vb;
        P3, ra;
        P4, aa;
    }

    PARAMETER_LISTS
    {
        VIEW_1, vl1;
        VIEW_2, vl2;
        /* ... */
    }
}

VARIABLE_LIST vl1
{
    MEMBERS
    {
        VL1, PARAM.P1;
        VL2, PARAM.P2;
        VL2, PARAM.P4;
    }
}

VARIABLE_LIST vl2 { /* ... */ }

VARIABLE va { /* ... */ }
VARIABLE vb { /* ... */ }
RECORD ra { /* ... */ }
ARRAY aa { /* ... */ }

RECORD cb1
{
    MEMBERS
    {
        /* ... */
        VIEWS_INDEX, __views_index; /* index of first view */
        /* ... */
    }
}

```

Figure 101 – Example EDD for a BLOCK_A with PARAMETER_LISTS

11 EDD development

11.1 Dictionaries

Using the dictionary file is convenient to separate localized text from other EDD content. Dictionaries containing strings e.g. of LABEL- or HELP-attributes, prompt strings of Builtin parameters.

The dictionary file can be updated separately from the individual EDD of each device.

If common dictionary entries exist, the EDD developer should choose them instead of defining individual strings in the device specific dictionary or in each device EDD.

Device EDD dictionaries do not overwrite common dictionary definitions. The EDD application shall use a common dictionary definition even if the EDD dictionary has the same entry.

11.2 Reserved

Reserved for other EDD development specifications.

Annex A

(normative)

Device simulation

Device simulation is an optional feature that is only for the EDD development to check EDD behavior. For this purpose some additional entry points will be used by a specific EDD testing tool. This tool runs the EDD in a field device simulator.

The `device_simulation_method` shall be an identifier of a METHOD that shall only be used by field device simulators. In device simulators, this method shall be run at the monotonic period specified by the `device_simulation_background_period` VARIABLE.

The `device_simulation_background_period` shall be an identifier of a VARIABLE that shall only be used by field device simulators for defining the execution period for the `device_simulation_method`. This VARIABLE shall be of CLASS LOCAL and TYPE FLOAT and its value should be set using the `DEFAULT_VALUE` attribute. Simulators shall execute background methods with a periodic accuracy of ± 10 ms. `device_simulation_background_period` shall default to 1,0 s if the VARIABLE does not exist or does not have a `DEFAULT_VALUE`.

Annex B (informative)

Predefined identifiers

Table B.1 provides predefined identifiers.

Table B.1 – Predefined identifiers

Item type	Identifier	Profile	Description
VARIABLE	comm_status	HART	When bits are set in this VARIABLE there has been a communication error
IMAGE	device_icon	HART	An IMAGE for a device
MENU	device_root_menu	All	See 4.3
MENU	device_root_menu*	FOUNDATION fieldbus	BLOCK_A based MENU optimized for PC based applications
VARIABLE	device_simulation_background_period	HART	See Annex A
METHOD	device_simulation_method	HART	See Annex A
VARIABLE	device_status	HART	A bit set in this VARIABLE indicates a potential problem in the device
MENU	diagnostics_root_menu	All	See 4.3
MENU	diagnostics_root_menu*	FOUNDATION fieldbus	BLOCK_A based MENU optimized for PC based applications
MENU	download_to_device_root_menu	All	See 9.6.4
MENU	download_variables	PROFIBUS, PROFINET	See 9.6.3
VARIABLE	extended_device_status	HART	A bit set in this VARIABLE indicates a potential problem in the device
ARRAY OF VARIABLE	factory_protection_array	HART	This lists VARIABLES that are not copied from one device instance into new instances.
VARIABLE	hart_functions	HART	Indicates EDD application special features
MENU	hh_device_root_menu	FOUNDATION fieldbus	Device level MENU optimized for handheld applications
MENU	hh_diagnostics_root_menu	FOUNDATION fieldbus	Device level MENU optimized for handheld applications
MENU	hh_process_variables_root_menu	FOUNDATION fieldbus	Device level MENU optimized for handheld applications
MENU	hot_key	HART	Short cut for handheld applications
ARRAY	loop_warning_variables	HART	Array of VARIABLES that could disrupt, or change the output loop current when they are sent to and stored in the device
MENU	maintenance_root_menu	All	See 4.3
MENU	*Menu_Top*	FOUNDATION fieldbus	BLOCK_A MENU optimized for handheld applications
MENU	Menu_Main_Maintenance ^a	PROFIBUS	MENU added to the menu bar of the EDD application
MENU	Menu_Main_Specialist ^a	PROFIBUS	MENU added to the menu bar of the EDD application
COLLECTION	no_download*	FOUNDATION fieldbus	Writable COLLECTION of BLOCK_A PARAMETERS that should not be part of a bulk download (e.g. parameters that require interaction with a device for calibration)
MENU	offline_root_menu	All	See 4.3
MENU	OnlineWindow_display ^a	PROFIBUS	MENU containing process variables

Item type	Identifier	Profile	Description
METHOD	post_send_configuration_method	HART	METHOD called after download
METHOD	pre_send_configuration_method	HART	METHOD called before download
MENU	process_variables_root_menu*	FOUNDATION fieldbus	BLOCK_A based MENU optimized for PC based applications
MENU	process_variables_root_menu ^a	All	See 4.3
COMMAND	read_additional_device_status	HART	COMMAND to read additional diagnostic information
VARIABLE	response_code	HART	This VARIABLE contains the response code for the command response received
MENU	root_menu	HART	See 4.2
VARIABLE	std_ResponseCode	PROFIBUS, PROFINET	Standard communication response for PROFIBUS and PROFINET
MENU	Table_Main_Maintenance ^a	PROFIBUS	A starting point for the explorer view of a device
MENU	Table_Main_Specialist ^a	PROFIBUS	A starting point for the explorer view of a device
MENU	upload_from_device_root_menu	All	See 9.6.3
MENU	upload_variables	HART, PROFIBUS, PROFINET	See 9.6.4
COLLECTION	upload_wanted*	FOUNDATION fieldbus	Read-only COLLECTION of BLOCK_A PARAMETERS that should be included in an offline dataset
Key: All: HART, FOUNDATION fieldbus, PROFIBUS, PROFINET *: prefix and postfix for identifiers			
^a These identifiers should not be used to write new EDDs.			

IECNORM.COM :: Click to view the full PDF of IEC 61804-4:2015

SOMMAIRE

AVANT-PROPOS	130
INTRODUCTION.....	132
1 Domaine d'application	133
2 Références normatives	133
3 Termes, définitions, termes abrégés, acronymes et conventions.....	134
3.1 Termes généraux et définitions	134
3.2 Termes et définitions relatifs aux appareils modulaires	134
3.3 Abréviations et acronymes	135
3.4 Conventions.....	135
4 Description de l'interface utilisateur EDDL	135
4.1 Vue d'ensemble	135
4.2 Conventions de menus pour les applications portatives	136
4.3 Conventions de menus pour les applications PC.....	136
4.3.1 Vue d'ensemble	136
4.3.2 Menus root en ligne	137
4.3.3 Menu root hors ligne	137
4.3.4 Exemple de structure de menu EDD	138
4.3.5 Interface utilisateur	143
4.4 Conteneurs et éléments contenus	146
4.4.1 Vue d'ensemble	146
4.4.2 Conteneurs	146
4.4.3 Eléments contenus	149
4.5 Règles de disposition.....	153
4.5.1 Vue d'ensemble.....	153
4.5.2 Règles de disposition pour WIDTH et HEIGHT.....	153
4.5.3 Règles de disposition pour les attributs COLUMNBREAK et ROWBREAK.....	154
4.5.4 Exemples de disposition	160
4.5.5 Interface utilisateur conditionnelle	169
4.6 Eléments graphiques	170
4.6.1 Vue d'ensemble	170
4.6.2 Graphique et diagramme	171
4.6.3 Attributs communs	171
4.6.4 CHART	172
4.6.5 GRAPH.....	180
4.6.6 AXIS	189
4.6.7 IMAGE.....	190
4.6.8 GRID	191
5 Description de données EDDL	193
5.1 Variables	193
5.1.1 TYPEs de VARIABLE	193
5.1.2 CLASSE de VARIABLE.....	194
5.1.3 ACTIONS de VARIABLE	194
5.2 Données d'appareil stockées dans l'application EDDL	195
5.2.1 Vue d'ensemble	195

5.2.2	FILE	195
5.2.3	LIST	197
5.3	Exposition des éléments de données en dehors de l'application EDD	204
5.4	Initialisation d'instances EDD	204
5.4.1	Vue d'ensemble	204
5.4.2	Prise en charge de l'initialisation	204
5.4.3	TEMPLATE	204
5.5	Mapping de modèle d'appareil	205
5.5.1	BLOCK_A	205
5.5.2	BLOCK_B	206
6	Programmation avec la METHOD EDDL et utilisation de Builtins	206
6.1	Builtin MenuDisplay	206
6.2	Division par zéro et valeurs flottantes non déterminées	209
6.2.1	Valeurs entières et non signées	209
6.2.2	Valeurs de virgule flottante	209
7	Appareils modulaires	209
7.1	Vue d'ensemble	209
7.2	Identification des EDD	210
7.3	Modèle d'objet d'instance	210
7.4	Configuration hors ligne	211
7.5	Configuration en ligne	211
7.6	Exemple d'appareil modulaire simple	211
7.6.1	Généralités	211
7.6.2	Exemple de fichier EDD distinct avec référencement EDD direct	212
7.6.3	Exemple de fichier EDD distinct avec référencement EDD par classification et interfaces	214
7.6.4	Exemple de fichier EDD	217
7.6.5	Exemple de combinaison d'appareils modulaires uniques et distincts	218
7.7	COMPONENT_RELATION	218
7.7.1	Généralités	218
7.7.2	Utilisation de l'attribut NEXT_COMPONENT	218
7.7.3	Utilisation des attributs REQUIRED_RANGES et ADDRESSING	218
7.8	Téléchargement montant et descendant pour les appareils modulaires	219
7.9	Diagnostic	219
7.10	Lecture de la topologie d'un appareil modulaire	220
7.10.1	SCAN	220
7.10.2	Type de module DETECT	222
7.11	Contrôle de configuration	222
8	Session d'édition	223
8.1	Gestion des données	223
8.1.1	Vue d'ensemble	223
8.1.2	Règles générales	226
8.1.3	Mise en cache des données pour les boîtes de dialogue et les fenêtres	226
8.1.4	Mise en cache pour les METHODS	227
8.2	Aspects de l'interface d'utilisateur des sessions d'édition	230
8.3	Rôles utilisateurs	231
9	Configuration hors ligne et en ligne	231
9.1	Vue d'ensemble	231
9.2	Ensemble de données hors ligne	231

9.3	Configuration hors ligne	232
9.4	Ensemble de données en ligne	232
9.5	Configuration en ligne	232
9.6	Téléchargement montant et descendant.....	232
9.6.1	Vue d'ensemble	232
9.6.2	Reprise après erreur.....	234
9.6.3	Procédure de téléchargement montant	234
9.6.4	Procédure de téléchargement descendant	236
10	Description de la communication EDDL	238
10.1	COMMAND	238
10.1.1	Généralités	238
10.1.2	OPERATION.....	238
10.1.3	TRANSACTION	239
10.1.4	Adressage des commandes	243
10.2	Analyse des données reçues de l'appareil.....	243
10.2.1	Généralités	243
10.2.2	Analyse des éléments de données complexes	244
10.2.3	FOUNDATION Fieldbus	244
10.2.4	HART	244
10.2.5	PROFIBUS et PROFINET.....	244
10.3	Modèle de communications FOUNDATION fieldbus	244
11	Développement d'EDD.....	249
11.1	Dictionnaires	249
11.2	Réservé	250
Annexe A (normative)	Simulation d'appareil	251
Annexe B (informative)	Identificateurs prédéfinis	252
Figure 1	Exemple de menus racine EDD.....	143
Figure 2	Exemple d'application EDD pour les diagnostics	143
Figure 3	Exemple d'application EDD pour les variables de processus	144
Figure 4	Exemple d'application EDD pour les variables principales	144
Figure 5	Exemple d'application EDD pour les caractéristiques de l'appareil relatives au processus	145
Figure 6	Exemple d'application EDD pour les caractéristiques de l'appareil	145
Figure 7	Exemple d'application EDD pour les caractéristiques de maintenance.....	146
Figure 8	Utilisation de COLLECTION MEMBERS dans les MENUS du STYLE GROUP	149
Figure 9	Affichage des bits uniques d'une variable BIT_ENUMERATED.....	150
Figure 10	Affichage des bits multiples d'une variable BIT_ENUMERATED	151
Figure 11	Exemple d'application EDD pour une variable de type BIT_ENUMERATED	151
Figure 12	Exemple de code source EDD pour la disposition des éléments qui dépassent.....	155
Figure 13	Disposition pour les éléments qui dépassent	155
Figure 14	Exemple de code source EDD pour la disposition des lignes partiellement remplies.....	156
Figure 15	Disposition pour les lignes partiellement remplies	156

Figure 16 – Exemple de code source EDD pour la disposition des lignes partiellement remplies.....	157
Figure 17 – Disposition pour les lignes partiellement remplies	157
Figure 18 – Exemple de code source EDD pour la disposition des éléments volumineux	158
Figure 19 – Disposition pour les éléments volumineux	158
Figure 20 – Exemple de code source EDD pour la disposition des colonnes dans un groupe superposé.....	159
Figure 21 – Disposition pour les colonnes en groupe superposé	159
Figure 22 – Exemple de code source EDD pour la disposition pour les colonnes avec GRAPHS en groupe superposé	160
Figure 23 – Disposition pour les colonnes avec GRAPHS en groupe superposé	160
Figure 24 – Exemple d'EDD pour un menu de présentation.....	161
Figure 25 – Exemple d'application EDD pour une fenêtre de présentation.....	161
Figure 26 – Exemple d'EDD qui utilise un COLUMNBREAK	161
Figure 27 – Exemple d'application EDD pour une fenêtre de présentation.....	162
Figure 28 – Exemple d'EDD pour une fenêtre de présentation	162
Figure 29 – Exemple d'application EDD pour une fenêtre de présentation.....	163
Figure 30 – Exemple d'EDD pour les graphiques et diagrammes en ligne	163
Figure 31 – Exemple d'application EDD pour un graphique en ligne.....	164
Figure 32 – Exemple d'EDD pour les graphiques et diagrammes pleine largeur	164
Figure 33 – Exemple d'application EDD pour un graphique pleine largeur.....	165
Figure 34 – Exemple d'EDD pour les conteneurs imbriqués	166
Figure 35 – Exemple d'application EDD pour les conteneurs imbriqués.....	166
Figure 36 – Exemple d'EDD pour les éléments EDIT_DISPLAY	167
Figure 37 – Exemple d'application EDD pour les éléments EDIT_DISPLAY.....	168
Figure 38 – Exemple d'EDD pour les images	168
Figure 39 – Exemple d'application EDD pour les images.....	169
Figure 40 – Attributs HEIGHT et WIDTH pour les CHART et les GRAPH.....	171
Figure 41 – Attribut EMPHASIS pour différencier une ou plusieurs SOURCES ou WAVEFORMS	172
Figure 42 – Exemple d'un diagramme avec une courbe dans une boîte de dialogue	174
Figure 43 – Exemple de diagramme avec deux SOURCES.....	175
Figure 44 – Affichage d'un exemple de diagramme avec deux SOURCES	176
Figure 45 – Exemple de diagramme avec trois barres horizontales	178
Figure 46 – Affichage d'un exemple de diagramme avec trois barres horizontales	178
Figure 47 – Exemple d'un diagramme dans une boîte de dialogue	180
Figure 48 – Le graphique et les éléments visuels.....	181
Figure 49 – Exemple de graphique.....	184
Figure 50 – Plusieurs axes utilisés.....	185
Figure 51 – Exemple d'EDD avec fonctions d'agrandissement et de défilement prises en charge par l'appareil	189
Figure 52 – Exemple d'EDD pour une IMAGE	190
Figure 53 – Exemple d'EDD pour une IMAGE avec attribut LINK	191
Figure 54 – Exemple d'EDD pour une GRID.....	192
Figure 55 – Résultat de l'exemple EDD	193

Figure 56 – Mauvaise utilisation d'une variable BIT_ENUMERATED	193
Figure 57 – Utilisation d'une variable ENUMERATED à la place d'une variable BIT_ENUMERATED	193
Figure 58 – Exemple de déclaration de fichier.....	196
Figure 59 – Exemple de comparaison des signatures d'une vanne.....	197
Figure 60 – Exemple d'une déclaration de fichier plus complexe	198
Figure 61 – Exemple d'examen des signaux radar stockés.....	199
Figure 62 – Exemple d'EDD qui insère, remplace ou compare des signaux radar.....	204
Figure 63 – Exemple d'utilisation de l'attribut TEMPLATE	205
Figure 64 – Exemple de BLOCK_A	206
Figure 65 – Exemple d'assistant	208
Figure 66 – Les différentes relations de module.....	211
Figure 67 – Composants et configuration possible des appareils modulaires.....	212
Figure 68 – Exemple de fichier EDD distinct avec référencement EDD direct.....	213
Figure 69 – Exemple d'EDD pour le module1	213
Figure 70 – Exemple d'EDD pour le module2	214
Figure 71 – Exemple d'EDD pour un appareil modulaire	215
Figure 72 – Exemple d'EDD pour le module1	216
Figure 73 – Exemple d'EDD pour le module2.....	216
Figure 74 – Exemple d'EDD pour le module2.....	218
Figure 75 – Utilisation de l'attribut NEXT_COMPONENT.....	218
Figure 76 – Utilisation de l'attribut REQUIRED_RANGES	218
Figure 77 – Ordre de téléchargement d'un appareil modulaire	219
Figure 78 – Exemple de METHOD SCAN.....	221
Figure 79 – Exemple de METHOD DETECT.....	222
Figure 80 – Exemple de CHECK_CONFIGURATION METHOD	223
Figure 81 – Mise en cache des données d'une session hors ligne.....	224
Figure 82 – Mise en cache des données d'une session en ligne.....	225
Figure 83 – Boîtes de dialogue secondaires ou sous-fenêtres qui utilisent un cache d'édition partagé.....	226
Figure 84 – Boîtes de dialogue secondaires ou sous-fenêtres qui utilisent des caches d'édition distincts.....	227
Figure 85 – Mise en cache de données pour les METHODS imbriquées	228
Figure 86 – Mise en cache de données pour une METHOD appelée depuis une boîte de dialogue.....	228
Figure 87 – Mise en cache des données pour une METHOD qui appelle une boîte de dialogue avec un cache d'édition	229
Figure 88 – Mise en cache de données pour une METHOD qui appelle une boîte de dialogue.....	229
Figure 89 – Flux de données pour un téléchargement à destination de l'appareil	233
Figure 90 – Flux de données pour un téléchargement en provenance de l'appareil	234
Figure 91 – Exemple de masque d'élément unique	240
Figure 92 – Exemple de mapping d'un masque d'élément unique.....	240
Figure 93 – Masques d'éléments multiples.....	241
Figure 94 – Exemple de mapping d'un masque d'éléments multiples.....	241

Figure 95 – Qualificatif INFO	242
Figure 96 – Qualificatif INDEX	242
Figure 97 – Qualificatifs INFO et INDEX	242
Figure 98 – Exemple d'appareil avec 2 définitions de BLOCK_A uniques	246
Figure 99 – Exemple d'EDD pour un appareil avec 2 définitions de BLOCK_A uniques	247
Figure 100 – Exemple de BLOCK_A avec PARAMETER_LISTS	248
Figure 101 – Exemple d'EDD pour un BLOCK_A avec PARAMETER_LISTS	249
Tableau 1 – Liste des identificateurs de menu root définis pour les appareils portatifs	136
Tableau 2 – Liste des identificateurs de menu root définis pour les appareils sur PC	136
Tableau 3 – Alternatives de redémarrage pour les menus root en ligne	137
Tableau 4 – Alternatives de redémarrage pour les menus root hors ligne	138
Tableau 5 – Eléments contenus admis et STYLES par défaut	147
Tableau 6 – Etendue et applicabilité de WIDTH et de HEIGHT	154
Tableau 7 – Formats d'image	191
Tableau 8 – Traitement de chaîne	194
Tableau 9 – Exemples de résultats à virgule flottante	209
Tableau 10 – Utilisations de l'attribut COMPONENT_PATH	210
Tableau 11 – Classifications de diagnostic	220
Tableau 12 – Builtins pour le contrôle du cache de méthode	230
Tableau 13 – Liste des identificateurs de menu de téléchargement montant	234
Tableau 14 – Liste des identificateurs de menu de téléchargement descendant	236
Tableau 15 – Mapping des communications PROFIBUS et PROFINET	239
Tableau B.1 – Identificateurs prédéfinis	252

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

**BLOCS FONCTIONNELS (FB) POUR LES PROCÉDÉS INDUSTRIELS ET
LE LANGAGE DE DESCRIPTION ÉLECTRONIQUE DE PRODUIT (EDDL) –****Partie 4: Interprétation EDD****AVANT-PROPOS**

- 1) La Commission Electrotechnique Internationale (IEC) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de l'IEC). L'IEC a pour objet de favoriser la coopération internationale pour toutes les questions de normalisation dans les domaines de l'électricité et de l'électronique. A cet effet, l'IEC – entre autres activités – publie des Normes internationales, des Spécifications techniques, des Rapports techniques, des Spécifications accessibles au public (PAS) et des Guides (ci-après dénommés "Publication(s) de l'IEC"). Leur élaboration est confiée à des comités d'études, aux travaux desquels tout Comité national intéressé par le sujet traité peut participer. Les organisations internationales, gouvernementales et non gouvernementales, en liaison avec l'IEC, participent également aux travaux. L'IEC collabore étroitement avec l'Organisation Internationale de Normalisation (ISO), selon des conditions fixées par accord entre les deux organisations.
- 2) Les décisions ou accords officiels de l'IEC concernant les questions techniques représentent, dans la mesure du possible, un accord international sur les sujets étudiés, étant donné que les Comités nationaux de l'IEC intéressés sont représentés dans chaque comité d'études.
- 3) Les Publications de l'IEC se présentent sous la forme de recommandations internationales et sont agréées comme telles par les Comités nationaux de l'IEC. Tous les efforts raisonnables sont entrepris afin que l'IEC s'assure de l'exactitude du contenu technique de ses publications; l'IEC ne peut pas être tenue responsable de l'éventuelle mauvaise utilisation ou interprétation qui en est faite par un quelconque utilisateur final.
- 4) Dans le but d'encourager l'uniformité internationale, les Comités nationaux de l'IEC s'engagent, dans toute la mesure possible, à appliquer de façon transparente les Publications de l'IEC dans leurs publications nationales et régionales. Toutes divergences entre toutes Publications de l'IEC et toutes publications nationales ou régionales correspondantes doivent être indiquées en termes clairs dans ces dernières.
- 5) L'IEC elle-même ne fournit aucune attestation de conformité. Des organismes de certification indépendants fournissent des services d'évaluation de conformité et, dans certains secteurs, accèdent aux marques de conformité de l'IEC. L'IEC n'est responsable d'aucun des services effectués par les organismes de certification indépendants.
- 6) Tous les utilisateurs doivent s'assurer qu'ils sont en possession de la dernière édition de cette publication.
- 7) Aucune responsabilité ne doit être imputée à l'IEC, à ses administrateurs, employés, auxiliaires ou mandataires, y compris ses experts particuliers et les membres de ses comités d'études et des Comités nationaux de l'IEC, pour tout préjudice causé en cas de dommages corporels et matériels, ou de tout autre dommage de quelque nature que ce soit, directe ou indirecte, ou pour supporter les coûts (y compris les frais de justice) et les dépenses découlant de la publication ou de l'utilisation de cette Publication de l'IEC ou de toute autre Publication de l'IEC, ou au crédit qui lui est accordé.
- 8) L'attention est attirée sur les références normatives citées dans cette publication. L'utilisation de publications référencées est obligatoire pour une application correcte de la présente publication.
- 9) L'attention est attirée sur le fait que certains des éléments de la présente Publication de l'IEC peuvent faire l'objet de droits de brevet. L'IEC ne saurait être tenue pour responsable de ne pas avoir identifié de tels droits de brevets et de ne pas avoir signalé leur existence.

La Norme internationale IEC 61804-4 a été établie par le sous-comité 65E: Les dispositifs et leur intégration dans les systèmes de l'entreprise, du comité d'études 65 de l'IEC: Mesure, commande et automation dans les processus industriels.

Cette première édition annule et remplace l'IEC TR 61804-4 parue en 2006. Cette édition constitue une révision technique.

Cette édition inclut les modifications techniques majeures suivantes par rapport à l'édition précédente:

- Alinéas modifiés:
 - Description de données EDDL
 - Programmation avec la METHOD EDDL et utilisation de Builtins

- Session d'édition
- Configuration hors ligne et en ligne
- Description de la communication EDDL
- Améliorations de l'alinéa Description de l'interface utilisateur EDDL

Le texte de cette norme est issu des documents suivants:

FDIS	Rapport de vote
65E/465/FDIS	65E/481/RVD

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

Cette publication a été rédigée selon les Directives ISO/IEC, Partie 2.

Une liste de toutes les parties de la série IEC 61804, publiées sous le titre général *Blocs fonctionnels (FB) pour les procédés industriels et le langage de description électronique de produit (EDDL)*, peut être consultée sur le site web de l'IEC.

Les futures normes de cette série porteront dorénavant le nouveau titre général cité ci-dessus. Le titre des normes existant déjà dans cette série sera mis à jour lors de la prochaine édition.

Le comité a décidé que le contenu de cette publication ne sera pas modifié avant la date de stabilité indiquée sur le site web de l'IEC sous "<http://webstore.iec.ch>" dans les données relatives à la publication recherchée. A cette date, la publication sera

- reconduite,
- supprimée,
- remplacée par une édition révisée, ou
- amendée.

IMPORTANT – Le logo "colour inside" qui se trouve sur la page de couverture de cette publication, indique qu'elle contient des couleurs qui sont considérées comme utiles à une bonne compréhension de son contenu. Les utilisateurs devraient, par conséquent, imprimer cette publication en utilisant une imprimante couleur.

INTRODUCTION

La présente partie de l'IEC 61804 a été développée à partir d'informations des consortiums FDI Cooperation LLC (Foundation™ Fieldbus¹, HART®² Communication Foundation (HCF), PROFIBUS™³ Nutzerorganisation e.V. (PNO)), OPC Foundation (OPCF) et FDT Group. La norme IEC 61804 porte le titre général "Blocs fonctionnels (FB) pour les procédés industriels et le langage de description électronique de produit (EDDL)".

Cette édition reflète un grand nombre des diverses règles définies par les différentes bases de communication, mais cette édition n'est pas actuellement une représentation complète de ces règles définies par chacune des bases de communication. Par conséquent, une application EDD et un développeur EED devront dépendre à la fois de la norme IEC 61804-4 et des documents respectifs des bases de communication (par exemple, les spécifications, les exigences en matière de test, les exemples de test) pour développer une application conforme qui sera répondre aux exigences d'enregistrement des bases.

L'évaluation de la conformité d'une application EDD est de la responsabilité des bases de communication respectives. En cas d'ambiguïté, les règles des bases de communication respectives appliquent.

La présente partie de l'IEC 61804

- contient une présentation de l'utilisation du langage EDDL;
- fournit des exemples qui illustrent l'utilisation des constructions EDDL;
- montre comment les cas d'utilisation sont réalisés; et
- montre l'interprétation correcte d'une application EDDL pour chaque exemple.

Cette partie de la norme IEC 61804 n'est pas une explication EDDL et n'est pas destinée à remplacer la spécification EDDL.

On trouve des instructions pour l'application EDD qui décrivent les actions qui seront réalisées sans la prescription de la technologie utilisée dans la mise en œuvre de l'hôte. Par exemple, la construction FILE décrit les données sauvegardées par l'application EDD pour le compte de l'EDD. La construction FILE ne spécifie pas la manière dont les données sont sauvegardées. L'application EDD peut utiliser une base de données, un fichier plat ou toute autre mise en œuvre de son choix.

¹ FOUNDATION™ Fieldbus est l'appellation commerciale de Fieldbus Foundation. Cette information est donnée à l'intention des utilisateurs de la présente norme et ne signifie nullement que l'IEC approuve ou recommande l'emploi exclusif du produit ainsi désigné. Des produits équivalents peuvent être utilisés s'il est démontré qu'ils conduisent aux mêmes résultats.

² HART® est une marque déposée de HART Communication Foundation. Cette information est donnée à l'intention des utilisateurs de la présente norme et ne signifie nullement que l'IEC approuve ou recommande l'emploi exclusif du produit ainsi désigné. Des produits équivalents peuvent être utilisés s'il est démontré qu'ils conduisent aux mêmes résultats.

³ PROFIBUS et PROFINET sont les appellations commerciales de PROFIBUS Nutzerorganisation e.V. Cette information est donnée à l'intention des utilisateurs de ce document et ne signifie nullement que l'IEC approuve ou recommande l'emploi exclusif du produit ainsi désigné. Des produits équivalents peuvent être utilisés s'il est démontré qu'ils conduisent aux mêmes résultats.

BLOCS FONCTIONNELS (FB) POUR LES PROCÉDES INDUSTRIELS ET LE LANGAGE DE DESCRIPTION ÉLECTRONIQUE DE PRODUIT (EDDL) –

Partie 4: Interprétation EDD

1 Domaine d'application

La présente partie de l'IEC 61804 définit l'interprétation EDD pour les applications EDD et les EDD pour assurer l'interopérabilité EDD. Le présent document est destiné à veiller à ce que les développeurs d'appareils de terrain utilisent systématiquement les constructions EDDL et que les applications EDD aient la même interprétation des EDD. Il complète la spécification EDDL pour promouvoir l'interopérabilité de l'application EDDL et améliorer la portabilité EDD entre les applications EDDL.

2 Références normatives

Les documents suivants sont cités en référence de manière normative, en intégralité ou en partie, dans le présent document et sont indispensables pour son application. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

IEC 61784-1, *Réseaux de communication industriels – Profils – Partie 1: Profils de bus de terrain*

IEC 61784-2, *Réseaux de communication industriels – Profils – Partie 2: Profils de bus de terrain supplémentaires pour les réseaux en temps réel basés sur l'ISO/IEC 8802-3*

IEC 61804-2, *Blocs fonctionnels (FB) pour les procédés industriels – Partie 2: Spécification du concept de FB*

IEC 61804-34, *Blocs fonctionnels (FB) pour les procédés industriels et le langage de description électronique de produit (EDDL) – Partie 3: Sémantique et syntaxe EDDL*

IEC 61804-55, *Blocs fonctionnels (FB) pour les procédés industriels et le langage de description électronique de produit (EDDL) – Partie 5: Bibliothèque de Built-in EDDL*

ISO/IEC 10918 (toutes les parties), *Technologies de l'information – Compression numérique et codage des images fixes de nature photographique*

ISO/IEC 15948, *Information technology – Computer graphics and image processing – Portable Network Graphics (PNG): Functional specification* (disponible en anglais seulement)

4 A publier.

5 A publier.

3 Termes, définitions, termes abrégés, acronymes et conventions

Pour les besoins du présent document, les termes et définitions donnés dans l'IEC 61804-3 ainsi que les suivants s'appliquent.

3.1 Termes généraux et définitions

3.1.1

développeur EDD

personne ou équipe qui développe un EDD

3.1.2

conteneur

éléments de l'interface utilisateur qui contiennent des éléments d'interface d'un autre utilisateur

Note 1 à l'article: Les conteneurs peuvent être des menus, fenêtres, boîtes de dialogue, tableaux, pages, groupes et autres conteneurs.

3.1.3

élément contenu

éléments de l'interface utilisateur qui peuvent être contenus dans des conteneurs

Note 1 à l'article: Les pièces contenues peuvent être des variables, méthodes, graphiques, diagrammes, tableaux, images, textes statiques.

3.1.4

développeur de l'appareil

personne ou équipe qui développe un appareil et un EDD qui décrit l'appareil

3.1.5

portatif

appareil dont la résolution d'affichage limitée restreint l'interface utilisateur d'applications EDD

3.2 Termes et définitions relatifs aux appareils modulaires

3.2.1

canal

connexion à un processus qui est mesuré ou contrôlé

3.2.2

composant

logiciel ou matériel contenu dans le concept d'appareil modulaire

Note 1 à l'article: Un composant ne peut pas fonctionner séparément d'un appareil modulaire hôte. Un composant peut prendre en charge un ou plusieurs types d'appareils modulaires.

3.2.3

interface

déclarations de base de constructions de base

Note 1 à l'article: Une interface définit toutes les parties publiques que les composants peuvent utiliser.

3.2.4

fenêtre modale

fenêtre enfant qui exige que les utilisateurs interagissent avec elle pour pouvoir revenir à l'exploitation de l'application parent et éviter ainsi de bloquer le workflow dans la fenêtre principale

3.2.5

appareil modulaire

appareil qui peut contenir une gamme de composants divers pour logiciels et/ou matériels

3.3 Abréviations et acronymes

CP	<i>Communication Profile</i> (Profil de communication)
CPF	<i>Communication Profile Family</i> (Famille de profils de communication)
EDD	<i>Electronic Device Description</i> (Description électronique de produit)
EDDL	<i>Electronic Device Description Language</i> (Langage de description électronique de produit)
PC	<i>Personal computer</i> (Ordinateur personnel)
PI	PROFIBUS et PROFINET International
PI PROFILE PA	Profil spécifique PI pour les appareils de terrain d'automatisation de processus

3.4 Conventions

Il existe certaines différences d'utilisation et d'interprétation d'EDDL en fonction du réseau de communication utilisé et du modèle de l'appareil. Les différents réseaux de communication utilisés dans cette partie de la norme IEC 61804 sont:

- HART (conformément à l'IEC 61784-1 CPF 9)
- FOUNDATION fieldbus (conformément à l'IEC 61784-1 CPF 1)
- PROFIBUS (conformément à l'IEC 61784-1 CPF 3)
- PROFINET (conformément à l'IEC 61784-2 CPF 3)

Les exemples d'EDD donnés dans la présente norme ne montrent que certaines parties de la mise en œuvre de l'EDD et sont incomplets.

Les mots-clés EDDL sont écrits en majuscules. Si plusieurs éléments de construction de base sont impliqués, le mot-clé est écrit en majuscules, qui sont suivies d'un "s" minuscule. Par exemple: VARIABLES.

Builtin écrit avec une majuscule fait référence à des fonctions spécifiées dans l'IEC 61804-5.

EXEMPLE Builtin MenuDisplay fait référence au Builtin appelé MenuDisplay spécifié dans l'IEC 61804-5.

4 Description de l'interface utilisateur EDDL

4.1 Vue d'ensemble

La plupart des applications EDD peuvent être caractérisées comme applications PC ou portatives. Au vu de la taille relativement petite d'un appareil portatif, les applications portatives ne peuvent afficher qu'un petit nombre d'informations à la fois. En revanche, les applications PC peuvent proposer une interface utilisateur qui a bien plus d'intérêt, en grande partie grâce à leur taille d'écran supérieure.

Pour prendre en charge les capacités des applications PC, la construction MENU a été étendue dans l'IEC 61804-3 par rapport aux définitions précédentes données dans l'IEC 61804-2. Etant donné les différences au niveau des interfaces utilisateurs des applications PC et des applications portatives, de nombreux appareils définiront par logique deux hiérarchies de MENU: la première pour les applications portatives et la seconde pour les

applications PC. Certains MENUs peuvent être utilisés dans les deux hiérarchies. La hiérarchie entière peut donc ne pas être spécifiée en double.

Différentes structures de menu sont possibles en fonction des différentes classes d'applications. La présente norme doit être utilisée pour créer des structures de menu dans un EDD qui soient interprétées de manière non équivoque par des applications. Afin d'assurer une l'interopérabilité au travers des applications, cette norme doit être appliquée.

4.2 Conventions de menus pour les applications portatives

Les applications EDD utilisent des menus spécifiques dans les EDD pour afficher l'interface utilisateur de l'appareil (voir Tableau 1). Les éléments de l'interface utilisateur peuvent en outre être décrits pour les appareils portatifs et pour les PCs en même temps. Pour les appareils portatifs, les chaînes peuvent avoir, en plus d'un code de langue, un code zz spécifique à leur pays afin de spécifier les chaînes plus courtes ou les images à résolution moindre (voir la norme IEC 61804-3).

Tableau 1 – Liste des identificateurs de menu root définis pour les appareils portatifs

Identificateur de menu	STYLE par défaut	Description succincte
root_menu	TABLE	Menu racine portatif pour les appareils HART. Il est obligatoire pour l'ensemble des EDD HART.
Menu_Top*	MENU	Le préfixe du menu racine portatif pour les BLOCK_A MENU_ITEM pour les appareils FOUNDATION fieldbus, par exemple Menu_Top_TB

4.3 Conventions de menus pour les applications PC

4.3.1 Vue d'ensemble

Les applications EDD utilisent des menus spécifiques dans les EDD pour afficher l'interface utilisateur de l'appareil. De tels menus sont définis pour le diagnostic, les variables de processus, les caractéristiques des appareils et la configuration hors ligne. Le Tableau 2 définit les identificateurs pour les différents menus racines avec ses STYLES par défaut. Le STYLE par défaut est utilisé par l'application EDD si l'attribut STYLE n'est pas défini dans le MENU. La colonne "Usage" du Tableau 2 définit comment l'application EDD possède un accès en ligne et hors ligne aux paramètres de l'appareil (voir Article 8).

Tableau 2 – Liste des identificateurs de menu root définis pour les appareils sur PC

Identificateur de MENU	STYLE par défaut	Utilisation	Description succincte
device_root_menu	MENU	En ligne	Affichage des caractéristiques de l'appareil pour configuration
diagnostic_root_menu	MENU	En ligne	Affichages des diagnostics
maintenance_root_menu	MENU	En ligne	Affichage des caractéristiques de maintenance
offline_root_menu	TABLE	Hors ligne	Configuration hors ligne
process_variables_root_menu	MENU	En ligne	Affichage des variables de processus

Pour les EDD HART, l'application EDD doit afficher le menu offline_root_menu avec un STYLE WINDOW par défaut si STYLE n'est pas défini.

Grâce à des sous-menus, PROFIBUS et PROFINET permettent de définir différents paramètres d'accès, via la définition d'ACCESS ONLINE ou d'ACCESS OFFLINE.

Il convient que les EDD contiennent les menus root en ligne du Tableau 2. Les menus root en ligne sont facultatifs pour PROFIBUS, PROFINET et FOUNDATION fieldbus. Le menu root hors ligne est facultatif pour FOUNDATION fieldbus. Les EDD HART doivent avoir tous les menus root du Tableau 2.

4.3.2 Menus root en ligne

4.3.2.1 Généralités

Si aucun des menus root en ligne n'existe, des points d'entrée spécifiques pour le profil de communication doivent être utilisés (voir Tableau 3).

Tableau 3 – Alternatives de redémarrage pour les menus root en ligne

Profil de communication	Description des alternatives de redémarrage
FOUNDATION fieldbus	Menus déclarés dans l'attribut BLOCK_A MENU_ITEMS, par exemple device_root_menu_aiblock. Si les menus du bloc ne sont pas définis, un MENU de style TABLE doit être généré à partir de l'attribut BLOCK_A PARAMETERS.
HART	root_menu
PROFIBUS, PROFINET	Menu_Main_Specialist s'il existe ou Menu_Main_Maintenance. Ces menus incluent des sous-menus en ligne et hors ligne.

4.3.2.2 Menu root de diagnostic

Le menu diagnostic_root_menu comprend des affichages qui montrent l'état de l'appareil, des informations détaillées sur le diagnostic et peuvent inclure des affichages graphiques qui illustrent, par exemple, une caractéristique de valeur.

4.3.2.3 Menu root de variable de processus

Le process_variable_root_menu comprend des affichages qui montrent les mesures du processus et établissent des points avec leur qualité et d'importantes informations pour les opérateurs de processus, comme les séries.

4.3.2.4 Menu root d'appareil

Le menu device_root_menu intègre les caractéristiques d'un appareil. Ces caractéristiques peuvent être divisées entre celles qui sont relatives au processus et celles qui sont spécifiques à l'appareil. Cette structure n'est pas exigée si le nombre de caractéristiques est trop petit pour être partagé. Les sous-menus qui représentent une structure sont admis dans le menu des caractéristiques de l'appareil. En cas de création de sous-menus, les menus dessous peuvent être divisés entre caractéristiques relatives au processus et caractéristiques spécifiques à l'appareil.

4.3.2.5 Menu root de maintenance

Il convient que maintenance_root_menu contienne des fonctionnalités pour maintenir l'appareil pendant la phase d'exécution et qu'il montre par exemple des informations sur la dernière inspection de maintenance.

4.3.3 Menu root hors ligne

Le menu offline_root_menu est une arborescence de menus comportant des éléments de données et des méthodes de configuration hors ligne, entre autres. Il contient, en particulier, l'ensemble des paramètres de l'appareil spécifiques aux applications et peut également contenir d'importantes variables en lecture seule et inscriptibles. Pour plus d'informations

concernant les paramètres spécifiques aux applications (voir 9.3). Le menu peut comporter des méthodes hors ligne, par exemple des assistants de configuration.

Si le menu root hors ligne n'existe pas, un point d'entrée spécifique pour le profil de communication doit être utilisé (voir Tableau 4).

Tableau 4 – Alternatives de redémarrage pour les menus root hors ligne

Profil de communication	Description des alternatives de redémarrage
FOUNDATION fieldbus	BLOCK_A PARAMETERS
HART	MENU upload_variables
PROFIBUS, PROFINET	Table_Main_Specialist s'il existe ou Table_Main_Maintenance

4.3.4 Exemple de structure de menu EDD

L'exemple d'EDD dans la Figure 1 comporte des menus supplémentaires qui peuvent être ajoutés à un EDD pour des applications PC. Ces menus sont ajoutés aux menus existants pour les applications. Cet exemple est spécifique à un appareil avec une interface conforme à HART. Les exemples pour les appareils avec une interface conforme aux appareils FOUNDATION fieldbus, PROFIBUS et PROFINET ressembleraient beaucoup à celui-ci.

```

MENU diagnostic_root_menu
{
    LABEL "Diagnostics";
    STYLE MENU;                                /* not required to define STYLE in this case, */
                                              /* because of default STYLE of the root menu */

    ITEMS
    {
        status_window,                        /* menu: style=window */
        self_test                            /* method */
    }
}

MENU status_window
{
    LABEL "Status";
    STYLE WINDOW;
    ITEMS
    {
        standard_diagnostics_page,           /* menu: style=page */
        devspec_diagnostics_page            /* menu: style=page */
    }
}

MENU standard_diagnostics_page
{
    LABEL "Standard";
    STYLE PAGE;
    ITEMS
    {
        device_status                        /* variable */
    }
}

MENU devspec_diagnostics_page
{
    LABEL "Device Specific";
    STYLE PAGE;
    ITEMS
    {
        xmtr_specific_status_1,             /* variable */
        xmtr_specific_status_2             /* variable */
    }
}

METHOD self_test
{
    LABEL "Self Test";
    DEFINITION
    {
        /* elided */
    }
}

MENU maintenance_root_menu
{
    LABEL "Maintenance";
    STYLE MENU;                                /* not required to define STYLE in this case, */
                                              /* because of default STYLE of the root menu */

    ITEMS
    {
        device_mode_dialog,                 /* menu: style=dialog */
        teach_in                            /* method */
    }
}

MENU device_mode_dialog
{
    LABEL "Device Mode";
    STYLE DIALOG;
    ITEMS
    {
        mode_page                            /* menu: style=page */
    }
}

MENU mode_page
{
    LABEL "Process Variables";

```

```

STYLE PAGE;
ITEMS
{
    transducer_group,          /* menu: style=group */
    function_group             /* menu: style=group */
}
}

MENU transducer_group
{
    LABEL "Transducer";
    STYLE GROUP;
    ITEMS
    {
        trans_target_mode,     /* variable */
        trans_actual_mode      /* variable */
    }
}

MENU function_group
{
    LABEL "Function";
    STYLE GROUP;
    ITEMS
    {
        func_target_mode,      /* variable */
        func_actual_mode       /* variable */
    }
}

METHOD teach_in
{
    LABEL "Teach-in";
    DEFINITION
    {
        /* elided */
    }
}

MENU process_variables_root_menu
{
    LABEL "Process Variables";
    STYLE MENU;                /* not required to define STYLE in this case, */
                                /* because of default STYLE of the root menu */
    ITEMS
    {
        overview_window,       /* menu: style=window */
        primary_vars_window    /* menu: style=window */
    }
}

MENU overview_window
{
    LABEL "Overview";
    STYLE WINDOW;
    ITEMS
    {
        process_vars_page      /* menu: style=page */
    }
}

MENU process_vars_page
{
    LABEL "Process Variables";
    STYLE PAGE;
    ITEMS
    {
        pressure_group,        /* menu: style=group */
        temperature_group      /* menu: style=group */
    }
}

MENU pressure_group
{
    LABEL "Pressure";
    STYLE GROUP;
    ITEMS

```

```

    {
        pv_digital_value,          /* variable */
        pv_upper_range_value,      /* variable */
        pv_lower_range_value       /* variable */
    }
}

MENU temperature_group
{
    LABEL "Temperature";
    STYLE GROUP;
    ITEMS
    {
        sv_digital_value,          /* variable */
        sv_upper_range_value,      /* variable */
        sv_lower_range_value       /* variable */
    }
}

MENU primary_vars_window
{
    LABEL "Primary Variables";
    STYLE WINDOW;
    ITEMS
    {
        pressure_chart_page,       /* menu: style=page */
        temperature_chart_page     /* menu: style=page */
    }
}

MENU pressure_chart_page
{
    LABEL "Pressure";
    STYLE PAGE;
    ITEMS
    {
        pressure_chart             /* chart */
    }
}

CHART pressure_chart
{
    /* elided */
}

MENU temperature_chart_page
{
    LABEL "Temperature";
    STYLE PAGE;
    ITEMS
    {
        temperature_chart         /* chart */
    }
}

CHART temperature_chart
{
    /* elided */
}

MENU device_root_menu
{
    LABEL "Device";
    STYLE MENU;                      /* not required to define STYLE in this case, */
                                      /* because of default STYLE of the root menu */

    ITEMS
    {
        process_related_window,    /* menu: style=window */
        device_specific_window,    /* menu: style=window */
        master_reset               /* method */
    }
}

MENU process_related_window
{
    LABEL "Process Related";
    STYLE WINDOW;

```

```

ITEMS
{
    identification_page,          /* menu: style=page */
    output_info_page             /* menu: style=page */
}

MENU identification_page
{
    LABEL "Identification";
    STYLE PAGE;
    ITEMS
    {
        tag,                      /* variable */
        manufacturer,             /* variable */
        device_type,              /* variable */
        device_revision,          /* variable */
        descriptor,               /* variable */
        message                   /* variable */
    }
}

MENU output_info_page
{
    LABEL "Output Information";
    STYLE PAGE;
    ITEMS
    {
        range_values_group,       /* menu: style=group */
        sensor_limits_group       /* menu: style=group */
    }
}

MENU range_values_group
{
    LABEL "Range Values";
    STYLE GROUP;
    ITEMS
    {
        pv_units,                 /* variable */
        pv_urv,                   /* variable */
        pv_lrv                    /* variable */
    }
}

MENU sensor_limits_group
{
    LABEL "Sensor Limits";
    STYLE GROUP;
    ITEMS
    {
        sensor_units,             /* variable */
        upper_sensor_limit,        /* variable */
        lower_sensor_limit         /* variable */
    }
}

MENU device_specific_window
{
    LABEL "Device Specific";
    STYLE WINDOW;
    ITEMS
    {
        identification_page,      /* menu: style=page */
        calibration_page          /* menu: style=page */
    }
}

MENU calibration_page
{
    LABEL "Calibration";
    STYLE PAGE;
    ITEMS
    {
        sensor_limits_group,       /* menu: style=group */
        sensor_trim_group          /* menu: style=group */
    }
}

```

```

}

MENU sensor_trim_group
{
    LABEL "Sensor Trim";
    STYLE GROUP;
    ITEMS
    {
        upper_sensor_trim_point, /* variable */
        lower_sensor_trim_point, /* variable */
        sensor_trim              /* method */
    }
}

METHOD master_reset
{
    LABEL "Master Reset";
    DEFINITION
    {
        /* elided */
    }
}

```

Figure 1 – Exemple de menus racine EDD

4.3.5 Interface utilisateur

4.3.5.1 Diagnostics

La Figure 2 montre un exemple d'application EDD pour les diagnostics.

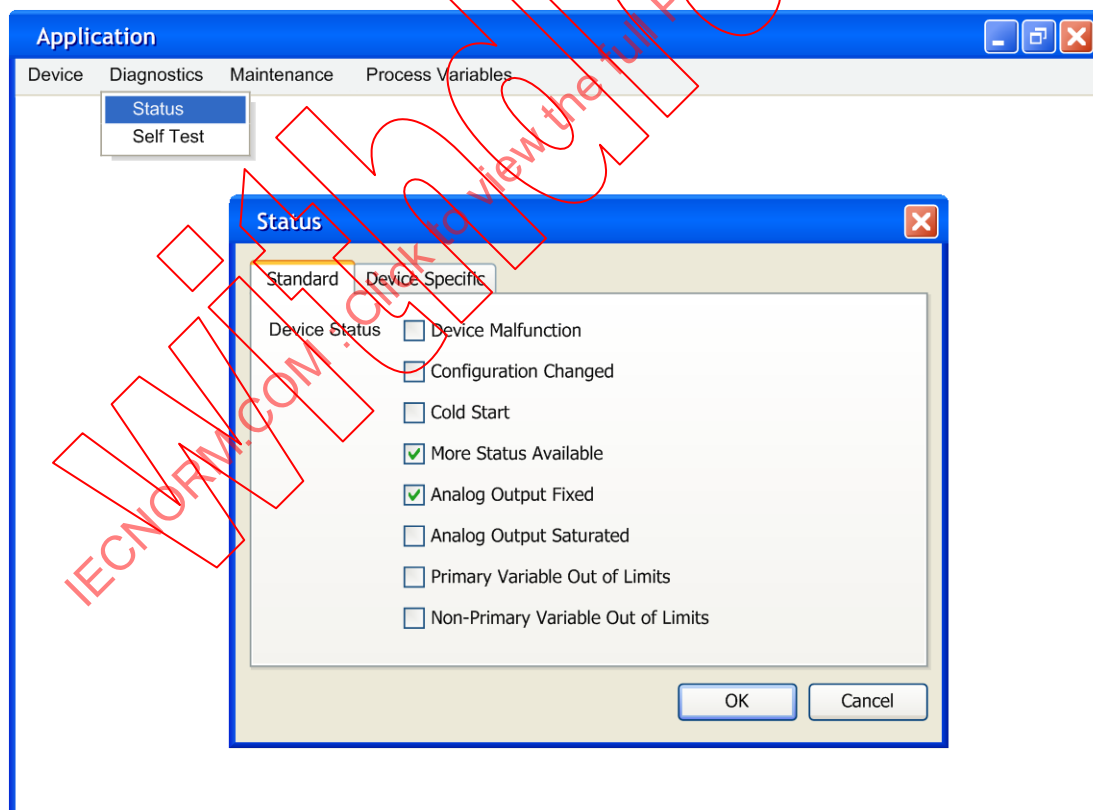


Figure 2 – Exemple d'application EDD pour les diagnostics

4.3.5.2 Variables de processus

La Figure 3 et la Figure 4 sont des exemples d'applications EDD de variables de processus.

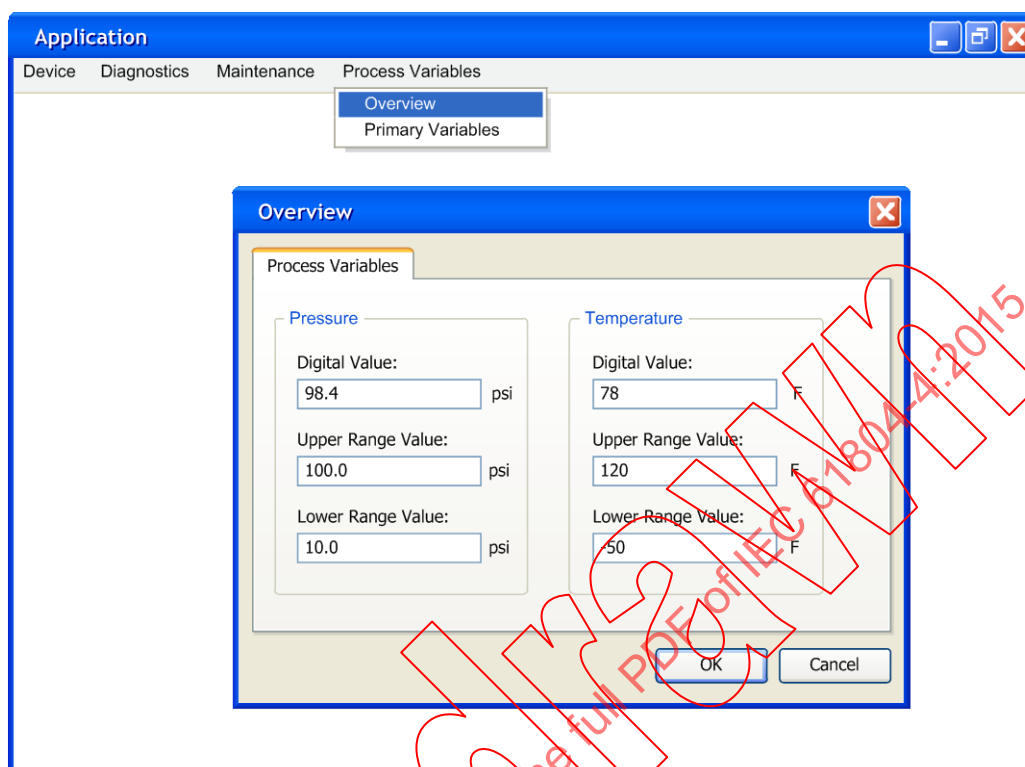


Figure 3 – Exemple d'application EDD pour les variables de processus

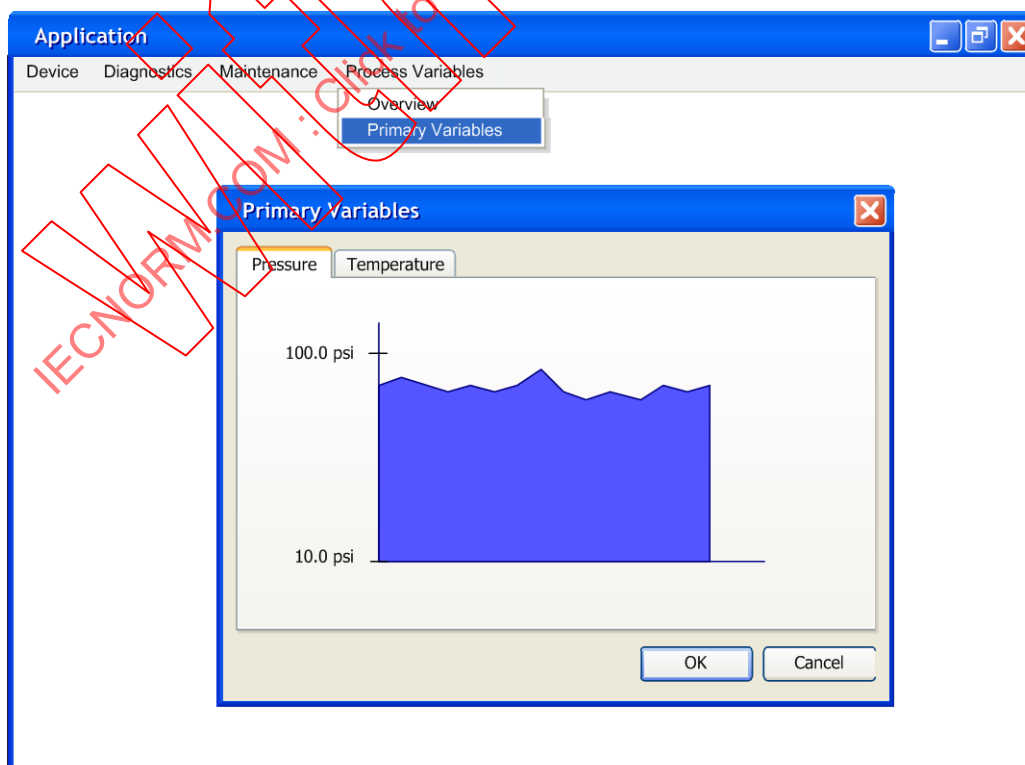


Figure 4 – Exemple d'application EDD pour les variables principales

4.3.5.3 Caractéristiques de l'appareil

La Figure 5, la Figure 6 et la Figure 7 sont des exemples d'applications EDD de caractéristiques de l'appareil.

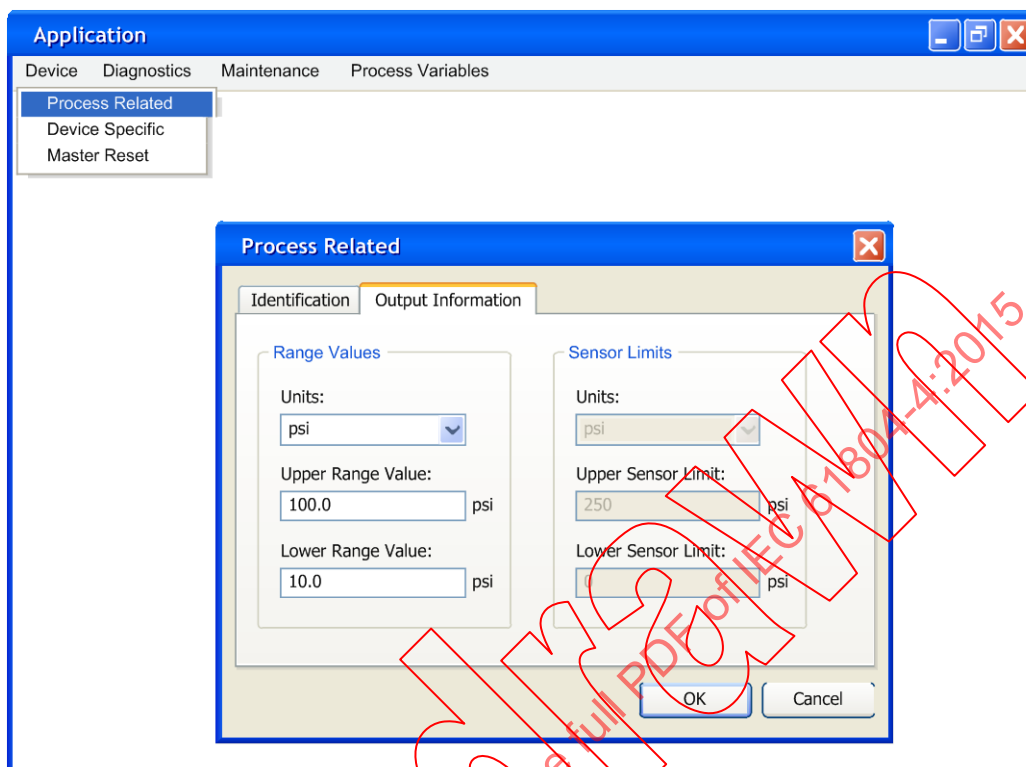


Figure 5 – Exemple d'application EDD pour les caractéristiques de l'appareil relatives au processus

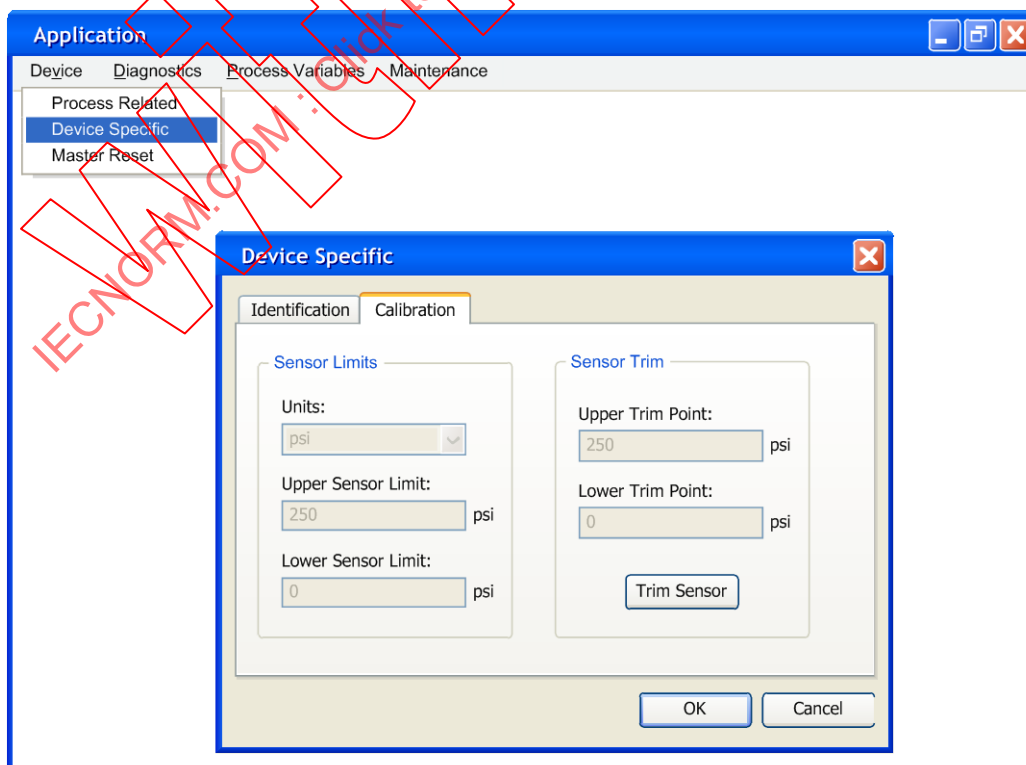


Figure 6 – Exemple d'application EDD pour les caractéristiques de l'appareil

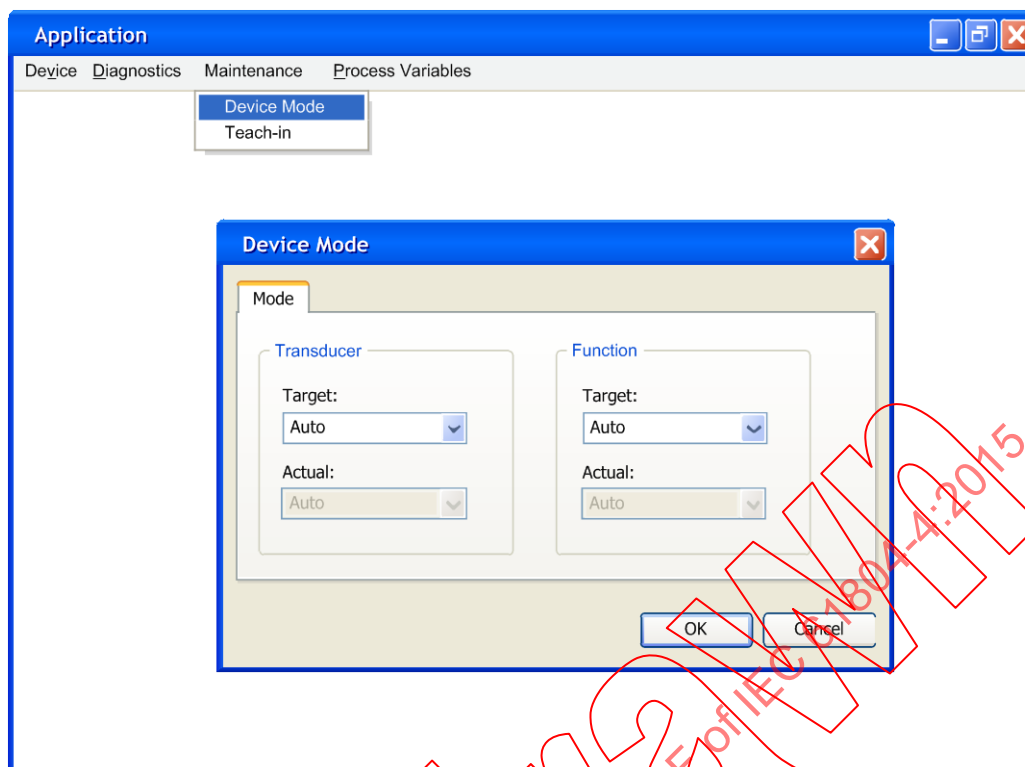


Figure 7 – Exemple d'application EDD pour les caractéristiques de maintenance

4.4 Conteneurs et éléments contenus

4.4.1 Vue d'ensemble

Les extensions de l'interface utilisateur sont basées sur un modèle simple d'interface utilisateur. Le modèle se décompose en deux concepts:

- les conteneurs; et
- les éléments contenus.

Les conteneurs sont appelés ainsi, car ils contiennent des éléments de l'interface d'un autre utilisateur. Les conteneurs peuvent comprendre les menus, fenêtres, boîtes de dialogue, tableaux, pages, groupes et d'autres conteneurs. Les conteneurs correspondent à un MENU. Ils se distinguent les uns des autres grâce à l'attribut STYLE. Cet attribut STYLE indique la manière dont le MENU sera affiché.

Les éléments contenus peuvent être des variables, méthodes, édition affichages, graphiques, diagrammes, images, texte statique, entre autres.

4.4.2 Conteneurs

4.4.2.1 STYLE admis et par défaut

Le Tableau 5 définit les éléments de l'interface utilisateur admis dans les conteneurs, les STYLES de substitution et par défaut ne sont pas définis. Si le style d'un menu n'est pas défini, un style par défaut peut être utilisé en fonction de l'emplacement du menu conteneur et du contenu du menu.

Le style par défaut d'un menu qui n'est pas contenu dans un menu adopte le style TABLE. Le menu HART root_menu est affiché comme un tableau de paramètres.

Règles générales:

- Si l'ACCESS en ligne ou hors ligne d'un MENU contenu est différent de l'accès du conteneur, l'application EDD doit utiliser STYLE WINDOW dans une fenêtre; sinon, elle doit utiliser STYLE DIALOG.

Tableau 5 – Eléments contenus admis et STYLES par défaut

Conteneur	Eléments contenus admis															STYLE par défaut des éléments contenus, utilisé si STYLE n'est pas défini
	MENU	WINDOW, DIALOG	PAGE	GROUP	TABLE	EDIT_DISPL AY	VARIABLE	COLLECTIO N RECORD	ARRAY, LIST	IMAGE, CHART, GRAPH, GRID	texte statique	METHOD	PLUGIN	COLUMNBR EAK, ROWBREAK	SEPARATO R	
MENU	X	X	D	D	2	X	7	2	2	2	2	X	X	6	6	MENU du STYLE MENU; si aucun élément de données n'est contenu dans le menu. DIALOG, si le menu comprend au moins un élément de données.
DIALOG, WINDOW	X	4	X	X	X	4	X	X	X	X	X	4	4	X	8	PAGE
PAGE	X	4	D	X	X	4	X	X	X	X	X	4	4	X	3	GROUP
GROUP	X	4	D	1	D	4	X	1	X	X	X	4	4	X	3	GROUP, si le parent du GROUP parent n'est pas dans le STYLE GROUP. DIALOG devient un bouton si le parent du GROUP parent est dans le STYLE GROUP.
TABLE	4	D	D	D	X	4	X	X	X	5	X	4	4	6	6	TABLE
Légende																
X Permis.																
D Le STYLE par défaut doit être utilisé (voir colonne "STYLE par défaut du sous-menu s'il n'est pas défini").																
1 Un seul sous-niveau est admis, par exemple GROUP peut contenir un GROUP, mais ce GROUP ne doit pas contenir plusieurs GROUPs, COLLECTIONs ou RECORDs.																
2 Doit être restitué dans un dialogue.																
3 Doit être interprété comme un COLUMNBREAK.																
4 Doit être restitué comme bouton ou lien hypertexte.																
5 Doit être restitué comme bouton, lien hypertexte ou en ligne.																
6 Doit être restitué comme une ligne ou ignoré.																
7 Doit être restitué comme bouton, lien hypertexte ou ignoré.																
8 SEPARATOR doit être traité comme un COLUMNBREAK.																

4.4.2.2 Menu

Un menu du STYLE MENU se présente sous forme d'un menu déroulant, menu contextuel ou d'une barre de navigation.

4.4.2.3 Fenêtre

Un menu de STYLE WINDOW prend l'apparence d'une fenêtre non modale.

4.4.2.4 Boîte de dialogue

Un MENU de STYLE DIALOG doit être une fenêtre modale. Si un dialogue contient une fenêtre subordonnée, l'application EDD doit gérer la fenêtre comme un sous-dialogue modal.

4.4.2.5 Tableau

Un menu de STYLE TABLE doit être restitué comme un tableau. Chaque élément doit être représenté sous forme d'une ou plusieurs lignes, par exemple les ARRAYS. Les sous-menus forment une hiérarchie, qui doit être affichée.

Il convient de ne définir l'attribut STYLE (STYLE = TABLE) que dans le menu racine d'une hiérarchie.

4.4.2.6 Page

Un menu de STYLE PAGE à l'intérieur d'un élément WINDOW ou DIALOG doit être restitué comme un onglet.

L'application EDD doit interpréter les ROWBREAK ou d'autres éléments entre les pages comme des séparations horizontales.

Les COLUMNBREAKs et SEPARATORs entre les pages doivent être ignorés.

4.4.2.7 Groupe

Un menu de STYLE GROUP prend l'apparence d'un cadre de groupe. Afin de maintenir une disposition épurée de l'interface utilisateur, les règles restrictives ci-dessous s'appliquent.

- Le niveau d'imbrication des déclarations du GROUP est réduit au nombre de deux. En d'autres termes, un GROUP peut contenir d'autres GROUPs, mais ces derniers ne doivent pas contenir davantage de GROUPs. Lorsqu'un GROUP de seconde couche contient un GROUP, ce GROUP doit être restitué soit en bouton, soit en lien hypertexte. Le LABEL doit apparaître sur le bouton ou en tant que lien hypertexte. Lorsque l'on appuie sur le bouton ou lien hypertexte, une boîte de dialogue correspondante doit s'ouvrir sur la fenêtre ou la boîte de dialogue en cours.

Au sein d'un GROUP, les méthodes et paramètres avec généralement un rapport logique sont regroupés. Le titre du GROUP indique cette relation, par exemple, "AnalogInput 1". La même chaîne est également souvent utilisée dans les libellés des éléments du GROUP, par exemple, "AnalogInput1_StaticRevision" ou "Analog Input1_Blockmode". Afin d'éviter cette duplication des informations, le développeur EDD peut réunir les éléments d'un GROUP dans une COLLECTION dans le but de remplacer les étiquettes originales. S'il doit être fait référence aux paramètres dans un GROUP, les paramètres peuvent être référencés avec la COLLECTION; dans d'autres cas, en particulier sans contexte sémantique supplémentaire, les paramètres peuvent être référencés directement.

La Figure 8 est un exemple d'EDD avec des COLLECTION MEMBERS au sein d'un MENU de STYLE GROUP.

```
VARIABLE ail_strev
{
    LABEL "Analog Input 1: Static Revision";
    TYPE INTEGER(4);
}

VARIABLE ail_bloMo
{
    LABEL "Analog Input 1: Block Mode";
    TYPE INTEGER(1);
}

MENU ail_group_double /* Leads to double display of */
{
    LABEL "Analog Input 1";
    STYLE GROUP;
    ITEMS
    {
        ail_strev,
        ail_bloMo
    }
}

COLLECTION ail_groupcol
{
    MEMBERS
    {
        strev,ail_strev,"Static Revision";
        bloMo,ail_bloMo,"Block Mode";
    }
}

MENU ail_group_single /* Leads to single display of */
{
    LABEL "Analog Input 1";
    STYLE GROUP;
    ITEMS
    {
        ail_groupcol.strev,
        ail_groupcol.bloMo
    }
}
```

Figure 8 – Utilisation de COLLECTION MEMBERS dans les MENUs du STYLE GROUP

4.4.3 Eléments contenus

4.4.3.1 Vue d'ensemble

La manière dont les éléments contenus sont restitués dans un conteneur est décrite dans le Paragraphe 4.4.3. Tous les conteneurs restituent les éléments contenus de la même façon. Les éléments contenus, comme les méthodes, les variables, entre autres, ont un attribut LABEL. Afin de ne pas perturber la disposition de l'application EDD, avec des boutons surdimensionnés pour les méthodes par exemple, il convient que le développeur EDD n'utilise pas de très longues chaînes pour les LABEL. Pour cette raison, certaines applications EDD peuvent tronquer ces libellés.

4.4.3.2 Méthodes

Il convient qu'une méthode soit restituée comme un bouton ou un lien hypertexte. Il convient que le LABEL de la METHOD correspondante apparaisse sur le bouton ou en tant que lien hypertexte. Lorsque l'on appuie sur le bouton ou lien hypertexte, il convient d'exécuter la METHOD correspondante.

4.4.3.3 Variables

4.4.3.3.1 Généralités

Il convient que l'étiquette, la valeur et, si définies, les unités de la variable soient affichées de manière cohérente avec la définition de la VARIABLE correspondante.

Le traitement général des variables se présente comme suit.

- HANDLING = READ ou l'attribut de l'élément du menu est READ_ONLY: il convient de ne pas éditer la valeur de la variable.
- CLASS = DYNAMIC: il convient de mettre constamment à jour la valeur avec les valeurs actuelles relevées sur l'appareil.

4.4.3.3.2 Variable de TYPE BIT_ENUMERATED

Les bits multiples peuvent être regroupés dans une variable énumérée à bit unique. Chaque bit pourrait avoir une signification distincte. Il est possible d'afficher chaque bit séparément et, de surcroît, d'afficher l'ensemble de la variable BIT_ENUMERATED. Les variables BIT_ENUMERATED peuvent être affichées sous forme de cases à cocher.

En cas d'affichage d'un bit de la variable BIT_ENUMERATED, il convient d'étendre la référence de variable avec un masque entre crochets et le LABEL de la variable n'est pas affiché.

La Figure 9 est un exemple d'EDD pour afficher les bits uniques d'une variable BIT_ENUMERATED.

```
VARIABLE var1
{
    LABEL "Var 1";
    TYPE BIT_ENUMERATED
    {
        { 0x1, "Bit 0"},
        { 0x2, "Bit 1"},
        { 0x4, "Bit 2"}
    }
}

MENU diagnostic_info
{
    LABEL "Diagnostic";
    ITEMS
    {
        var1[0x1],    // Bit 0
        var1[0x2],    // Bit 1
        var1[0x4]     // Bit 2
    }
}
```

Figure 9 – Affichage des bits uniques d'une variable BIT_ENUMERATED

La Figure 10 est un exemple d'EDD avec les différences entre l'affichage complet d'une variable BIT_ENUMERATED et la façon dont elle peut s'afficher si tous les bits sont adressés de manière explicite.

```

VARIABLE var1
{
    LABEL "Var 1";
    TYPE BIT_ENUMERATED
    {
        { 0x1, "Bit 0"},
        { 0x2, "Bit 1"},
        { 0x4, "Bit 2"}
    }
}

MENU var1_group
{
    LABEL var1.LABEL;
    STYLE GROUP;
    ITEMS
    {
        var1[0x1],    // Bit 0
        var1[0x2],    // Bit 1
        var1[0x4]     // Bit 2
    }
}

MENU diagnostic_info
{
    LABEL "Diagnostic";
    STYLE PAGE;
    ITEMS
    {
        var1,          // all bit should be shown
        COLUMNBREAK,
        var1[0x1],      // Bit 0
        var1[0x2],      // Bit 1
        var1[0x4],      // Bit 2
        COLUMNBREAK,
        var1_group
    }
}

```

Figure 10 – Affichage des bits multiples d'une variable BIT_ENUMERATED

La Figure 11 montre de quelle manière peut apparaître le résultat de l'exemple d'EDD de la Figure 10 dans une application EDD.

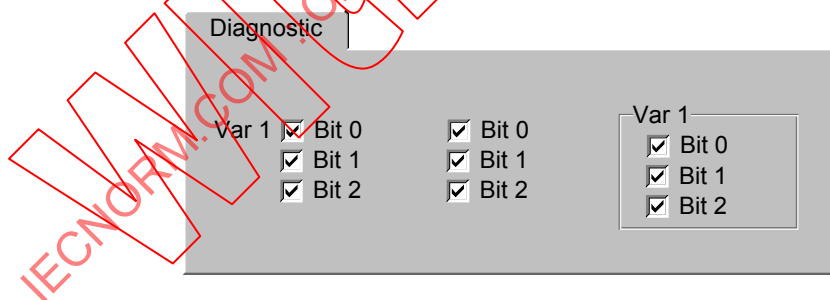


Figure 11 – Exemple d'application EDD pour une variable de type BIT_ENUMERATED

4.4.3.3.3 Variable de TYPE INDEX

Quand une VARIABLE de type INDEX est présentée à l'utilisateur, l'application EDD doit déterminer le texte qui doit être affiché via les règles suivantes:

- 1) si la description dans les ELEMENTS référencés existe, cela doit être affiché, sinon
- 2) si le LABEL de la variable existe, cela doit être affiché, sinon
- 3) le LABEL de la matrice doit être affiché avec la valeur numérique d'INDEX, par exemple myarray[3].

Si la valeur INDEX n'est pas dans la plage de la matrice relative, il convient qu'un texte informe l'utilisateur que la valeur n'est pas dans la plage.

Si la variable est éditable, il convient de la présenter sous forme de boîte combinée.

4.4.3.4 Enregistrements et collections

Les RECORDs et COLLECTIONs doivent être affichés sous forme de MENU avec un STYLE GROUP. Les COLLECTIONs peuvent contenir des COLLECTIONs. Elles sont traitées comme des groupes imbriqués (voir 4.4.2.7). Les règles au sujet des groupes imbriqués s'appliquent également dans ce cas.

4.4.3.5 Matrices et listes

Les matrices de valeur, les matrices de référence et les LISTs doivent être restituées sous forme d'une zone à faire défiler à l'intérieur du conteneur. Le LABEL de l'élément doit toujours être visible.

NOTE HART ne supporte pas les références de matrices et listes dans les conteneurs.

4.4.3.6 Edition des affichages

Un EDIT_DISPLAY doit être rendu dans le conteneur sous forme de bouton ou de lien hypertexte. Le LABEL de l'EDIT_DISPLAY correspondant doit apparaître sur le bouton ou en tant que lien hypertexte.

Quand le bouton ou le lien hypertexte est ouvert, l'EDIT_DISPLAY correspondant doit s'ouvrir dans une fenêtre modale. Un EDIT_DISPLAY contient des DISPLAY_ITEMS et des EDIT_ITEMS. L'application EDD doit afficher les éléments DISPLAY_ITEMS sur les éléments EDIT_ITEMS. Les DISPLAY_ITEMS doivent être affichés en tant qu'élément en lecture seule et les EDIT_ITEMS doivent être gérés comme ITEMS depuis un MENU de STYLE DIALOG.

4.4.3.7 Graphiques

Il convient d'afficher un graphique de manière cohérente avec la définition du GRAPH correspondant. Se reporter aux règles de disposition définies en 4.5 et dans l'IEC 61804-3 pour obtenir des informations sur l'influence de l'attribut WIDTH sur la disposition du GRAPH.

4.4.3.8 Diagrammes

Il convient d'afficher un diagramme de manière cohérente avec la définition du CHART correspondant. Se référer aux règles de disposition définies dans le Paragraphe 4.5 et dans la norme IEC 61804-3 pour obtenir des informations sur l'influence de l'attribut WIDTH sur la disposition du CHART.

4.4.3.9 Images

Il convient d'afficher les images référencées par le MENU.

L'application EDD doit afficher une IMAGE dans sa taille d'origine, si elle n'est pas référencée avec un qualificateur INLINE d'élément de menu. L'application EDD doit insérer un ROWBREAK sur l'IMAGE si des éléments existent avant l'IMAGE; elle doit insérer un ROWBREAK sous l'IMAGE si des éléments existent après l'IMAGE.

L'application EDD doit réduire la taille d'une IMAGE avec un élément de menu de qualificateur INLINE si la largeur de l'IMAGE est supérieure à celle de la colonne où l'IMAGE se trouve.

L'application EDD ne doit pas augmenter la taille d'une IMAGE.

L'application EDD doit maintenir le facteur de forme d'origine des IMAGES.

4.4.3.10 Texte statique

Il convient d'afficher le texte référencé par le MENU. Le texte sera réparti sur plusieurs lignes s'il est plus long que la largeur de la boîte de dialogue, de la fenêtre, de la page, du groupe ou de la colonne.

4.4.3.11 Grille

Le LABEL de la grille s'affiche en premier, puis la zone de la grille avec la largeur de la boîte dialogue, de la fenêtre, de la page, du groupe ou de la colonne apparaît en dessous, ainsi que la hauteur nécessaire pour afficher les données ou ce qui s'affiche à l'écran. Les barres de défilement sont utilisées pour faire défiler la zone de la grille si la largeur ou la hauteur est trop petite pour afficher l'ensemble des données.

4.5 Règles de disposition

4.5.1 Vue d'ensemble

Les règles de disposition sont des règles que l'application EDD doit appliquer pour l'agencement du contenu des fenêtres et boîtes de dialogue affichées à l'écran.

Chaque conteneur définit le cadre englobant du conteneur. Il s'agit de l'emplacement du conteneur dans lequel son contenu doit être affiché. Par exemple, le cadre englobant d'une fenêtre ou d'une boîte de dialogue correspond à toute la fenêtre dont sont soustraites la bordure et la barre de titre.

Les ITEMS d'un conteneur doivent être affichés dans l'ordre dans lequel ils apparaissent dans l'EDD. Il convient d'organiser le contenu du conteneur en commençant par la zone située à gauche du conteneur. Les éléments doivent être affichés verticalement du conteneur. Lorsque l'on rencontre un COLUMNBREAK, une nouvelle colonne doit être créée. Les éléments suivants doivent être affichés dans la prochaine colonne en commençant par le haut du conteneur. Ce processus continue jusqu'à la fin de l'affichage de tous les éléments. Les sauts de colonne doivent uniquement être introduits lorsque l'on rencontre un COLUMNBREAK. L'application EDD ne doit pas introduire de saut de colonne de sa propre initiative.

Si le texte statique contient des retours chariot et sauts de ligne, il convient alors de le répartir sur plusieurs lignes s'il est long.

L'ordre des éléments dans les listes de cause ou d'effet des relations d'unité et d'actualisation n'est pas utilisé à des fins d'affichage. Si une relation WRITE_AS_ONE rend nécessaire de terminer les éléments affichés, ces éléments supplémentaires doivent apparaître dans l'ordre de la relation WRITE_AS_ONE.

4.5.2 Règles de disposition pour WIDTH et HEIGHT

VARIABLE, CHART, GRAPH et GRID possèdent les attributs WIDTH et HEIGHT. L'attribut WIDTH définit le nombre de colonnes à afficher. L'attribut HEIGHT définit la taille verticale de l'affichage en tant que multiple de la hauteur minimale d'un champ d'entrée normal (voir Tableau 6). La valeur par défaut de WIDTH et de HEIGHT est MEDIUM pour CHART, GRAPH, GRID; elle est XXX_SMALL pour la largeur et la hauteur de VARIABLE.

Il convient de diviser la largeur de l'application EDD jusqu'à 5 colonnes en fonction des valeurs COLUMNBREAK et WIDTH des conteneurs. Les colonnes qui atteindraient une colonne supérieure à la 5^e colonne doivent être déplacées vers la ligne suivante via un ROWBREAK implicite.

Un ROWBREAK implicite a le même comportement qu'un ROWBREAK explicite défini.

Les attributs WIDTH et HEIGHT d'un conteneur définissent le nombre de colonnes que le conteneur peut comprendre et la hauteur en multiples de la hauteur minimale possible pour un champ simple, par exemple une variable, une chaîne de constance, un menu et une référence de méthode (voir Tableau 6).

Tableau 6 – Etendue et applicabilité de WIDTH et de HEIGHT

Valeurs WIDTH et HEIGHT	WIDTH	HEIGHT	Applicabilité
XXX_SMALL	1	1	VARIABLE
XX_SMALL	1	3	CHART GRAPH GRID VARIABLE
X_SMALL	1	5	CHART GRAPH GRID VARIABLE
SMALL	2	7	CHART GRAPH GRID VARIABLE
MEDIUM	3	8	CHART GRAPH GRID VARIABLE
LARGE	4	9	CHART GRAPH GRID VARIABLE
X_LARGE	5	11	CHART GRAPH GRID VARIABLE
XX_LARGE	5	13	CHART GRAPH GRID VARIABLE

4.5.3 Règles de disposition pour les attributs COLUMNBREAK et ROWBREAK

4.5.3.1 Disposition pour les éléments qui dépassent

L'application EDD doit insérer l'ensemble des éléments du menu dans une seule colonne, les uns à la suite des autres, jusqu'à rencontrer un COLUMNBREAK. Le COLUMNBREAK déclenchera l'insertion du prochain élément du menu dans la plus haute ligne de la colonne suivante. Si l'un des éléments du menu occupe une zone dans la prochaine colonne, alors l'élément du menu sera inséré dans la ligne immédiatement en dessous de l'élément du menu occupé. Voir la Figure 12 pour un exemple de code source EDD et la Figure 13 pour la disposition correspondante.

```

MENU protruding_elements
{
    LABEL "protruding_elements";
    STYLE WINDOW;
    ITEMS
    {
        Element1,
        Element2,
        Element3,
        Element4,
        SmallGraph5,      //WIDTH SMALL (2 columns), HEIGHT SMALL (7 height units)
        Element6,
        Element7,
        Element8,
        COLUMNBREAK,
        Element9,
        Element10,
        Element11,
        COLUMNBREAK,
        Element12,
        Element13,
        Element14,
    }
}
    
```

Figure 12 – Exemple de code source EDD pour la disposition des éléments qui dépassent

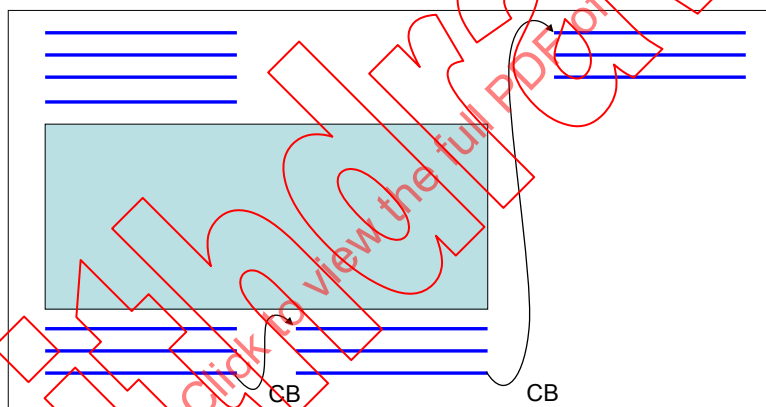


Figure 13 – Disposition pour les éléments qui dépassent

4.5.3.2 Disposition pour les lignes partiellement remplies

L'application EDD doit insérer l'ensemble des éléments du menu dans une seule colonne, les uns à la suite des autres, jusqu'à rencontrer un ROWBREAK. Un ROWBREAK déclenchera l'insertion du prochain élément du menu dans la ligne la plus basse de la 1^{ère} colonne (de la même manière qu'un retour chariot). Le ROWBREAK doit également mettre en place une barrière horizontale afin que tout futur COLUMNBREAK empêche l'insertion d'un élément du menu sur cette ligne. Voir la Figure 14 pour un exemple de code source EDD et la Figure 15 pour la disposition correspondante.

```

MENU Fig13
{
    LABEL "Fig13";
    STYLE WINDOW;
    ITEMS
    {
        Element1,
        Element2,
        Element3,
        COLUMNBREAK,
        Element4,
        Element5,
        Element6,
        Element7,
        ROWBREAK,
        SmallGraph8, //WIDTH SMALL, HEIGHT SMALL, occupies 2 columns
        Element9,
        Element10,
        Element11,
        COLUMNBREAK,
        Element12,
        Element13,
        Element14,
        COLUMNBREAK,
        Element15,
        Element16,
        Element17,
    }
}
    
```

Figure 14 – Exemple de code source EDD pour la disposition des lignes partiellement remplies

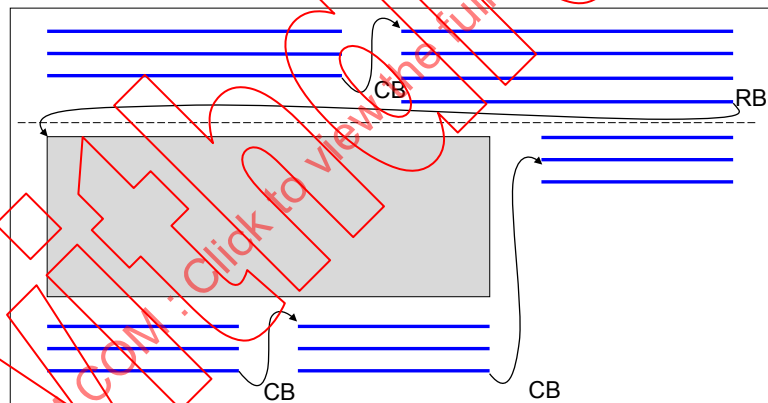


Figure 15 – Disposition pour les lignes partiellement remplies

La Figure 17 montre que les éléments de menus dans les lignes avec deux colonnes (lignes 1 et 3) ont étendu l'espace de cellule disponible, de telle sorte que la ligne prend maintenant exactement la même largeur que les lignes avec trois colonnes. La Figure 16 montre un exemple du code source EDD correspondant.

```

MENU partially_filled_rows
{
    LABEL "partially filled rows";
    STYLE WINDOW;
    ITEMS
    {
        Element1,
        Element2,
        COLUMNBREAK,
        Element3,
        Element4,
        ROWBREAK,
        Element5,
        Element6,
        COLUMNBREAK,
        Element7,
        Element8,
        COLUMNBREAK,
        Element9,
        Element10,
        ROWBREAK,
        Element11,
        Element12,
        COLUMNBREAK,
        Element13,
        Element14
    }
}

```

Figure 16 – Exemple de code source EDD pour la disposition des lignes partiellement remplies

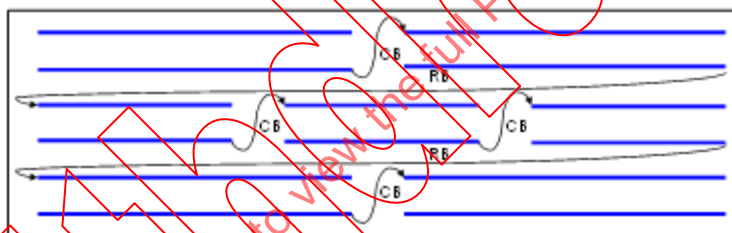


Figure 17 – Disposition pour les lignes partiellement remplies

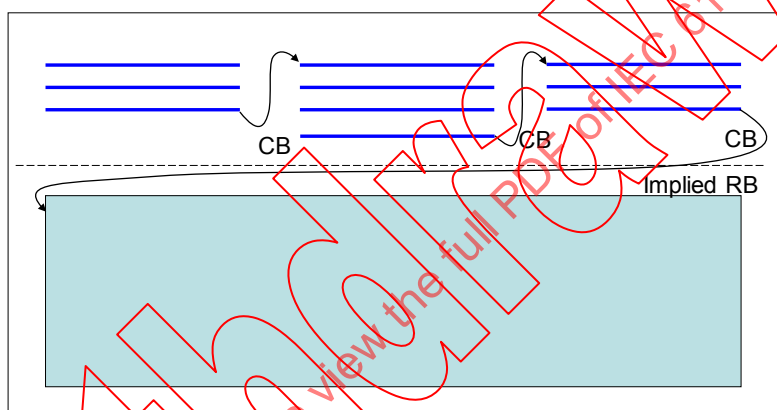
4.5.3.3 Disposition pour les éléments surdimensionnés

L'application EDD doit insérer l'ensemble des éléments du menu dans une seule colonne, les uns à la suite des autres, jusqu'à rencontrer un COLUMNBREAK. Le COLUMNBREAK déclenchera l'insertion du prochain élément du menu dans la plus haute ligne de la colonne suivante. Si l'un des éléments du menu à insérer est plus large que le nombre de colonnes disponibles, l'élément du menu sera par conséquent inséré dans la ligne la plus basse de la 1^{ère} colonne (c'est-à-dire un ROWBREAK implicite). Voir la Figure 18 pour un exemple de code EDD et la Figure 19 pour la disposition correspondante.

```

MENU layout_for_oversized_elements
{
    LABEL "layout for oversized elements";
    STYLE WINDOW;
    ITEMS
    {
        Element1,
        Element2,
        Element3,
        COLUMNBREAK,
        Element4,
        Element5,
        Element6,
        COLUMNBREAK,
        Element7,
        Element8,
        Element9,
        COLUMNBREAK, //Results in implied ROWBREAK
        MediumGraph10 //WIDTH MEDIUM, HEIGHT MEDIUM, occupies 3 columns
    }
}
    
```

Figure 18 – Exemple de code source EDD pour la disposition des éléments volumineux



Anglais	Français
Implied RB	RB implicite

Figure 19 – Disposition pour les éléments volumineux

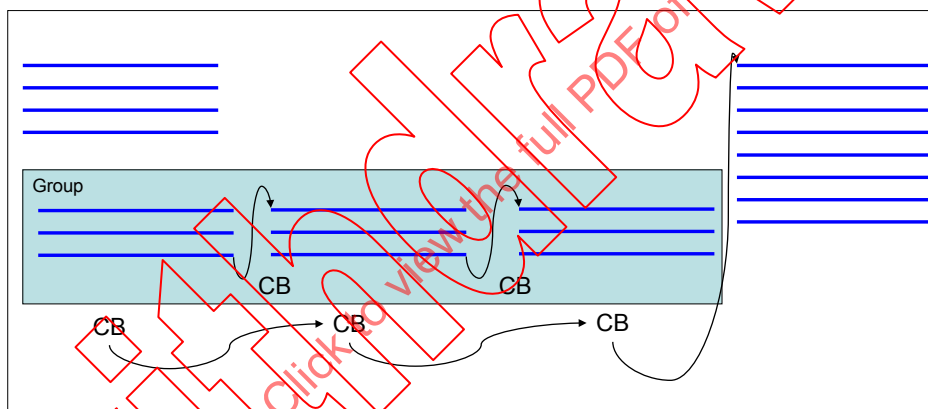
4.5.3.4 Disposition pour les colonnes en groupe superposé

L'application EDD doit insérer l'ensemble des éléments du menu dans une seule colonne, les uns à la suite des autres, jusqu'à rencontrer un COLUMNBREAK. Le COLUMNBREAK déclenchera l'insertion du prochain élément du menu dans la plus haute ligne de la colonne suivante. Si l'élément du menu se trouve dans un menu imbriqué (comme un GROUP), l'élément restera alors dans le menu parent. Voir la Figure 20 pour un exemple de code source EDD et la Figure 21 pour la disposition correspondante.

```

MENU layout_for_columns_in_stacked_group
{
    LABEL "layout for columns in stacked group";
    STYLE WINDOW;
    ITEMS
    {
        Element1,
        Element2,
        Element3,
        Element4,
        GroupWithThreeColumns5,
        COLUMNBREAK,
        COLUMNBREAK,
        COLUMNBREAK,
        Element6,
        Element7,
        Element8,
        Element9,
        Element10,
        Element11,
        Element12,
        Element13
    }
}
    
```

Figure 20 – Exemple de code source EDD pour la disposition des colonnes dans un groupe superposé



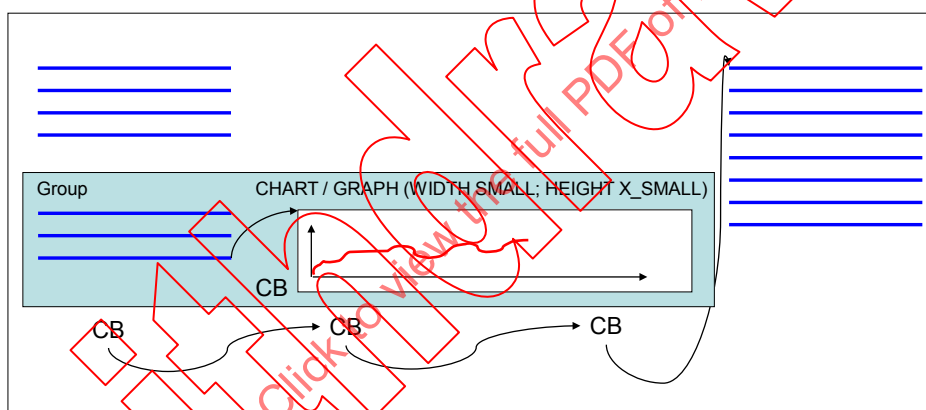
Anglais	Français
Group	Groupe

Figure 21 – Disposition pour les colonnes en groupe superposé

Les graphiques dans des groupes doivent être traités de manière équivalente comme plusieurs colonnes dans un groupe imbriqué. Voir la Figure 22 pour un exemple de code EDD et la Figure 23 pour la disposition correspondante.

```
MENU layout_for_columns_in_stacked_group_with_a_graph
{
    LABEL "layout for columns in stacked group with a graph";
    STYLE WINDOW;
    ITEMS
    {
        Element1,
        Element2,
        Element3,
        Element4,
        GroupWithTreeElementsOneColumnsBreakChartWidthSmall,
        COLUMNBREAK,
        COLUMNBREAK,
        COLUMNBREAK,
        Element8,
        Element9,
        Element10,
        Element11,
        Element12,
        Element13,
        Element14,
        Element15
    }
}
```

Figure 22 – Exemple de code source EDD pour la disposition pour les colonnes avec GRAPHS en groupe superposé



Anglais	Français
Group	Groupe

Figure 23 – Disposition pour les colonnes avec GRAPHS en groupe superposé

4.5.4 Exemples de disposition

4.5.4.1 Disposition pour une seule colonne

La plus simple des dispositions consiste en une liste de variables, méthodes, affichages d'édition, texte statique, graphiques et diagrammes.

```

MENU overview_window
{
    LABEL "Overview";
    STYLE WINDOW;
    ITEMS
    {
        primary_variable,          /* variable */
        upper_range_value,         /* variable */
        lower_range_value,         /* variable */
        master_reset,              /* method */
        self_test                  /* method */
    }
}

```

Figure 24 – Exemple d'EDD pour un menu de présentation

L'EDD représentée dans la Figure 24 affiche les éléments verticalement dans une fenêtre. Les éléments sont affichés en commençant par le côté gauche de l'écran et utilisent tout l'espace qui leur est nécessaire dans la fenêtre (voir Figure 25).

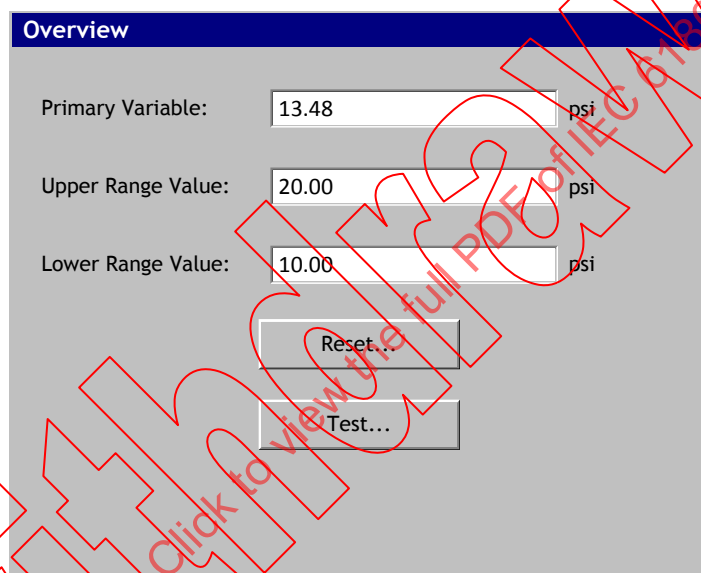


Figure 25 – Exemple d'application EDD pour une fenêtre de présentation

4.5.4.2 Disposition pour plusieurs colonnes et lignes

Plusieurs colonnes de variables, méthodes, groupes, images, graphiques et diagrammes peuvent également être utilisées (voir Figure 26).

```

MENU overview_page
{
    LABEL "Overview";
    STYLE PAGE;
    ITEMS
    {
        upper_range_value,          /* variable */
        lower_range_value,         /* variable */
        upper_sensor_limit,         /* variable */
        lower_sensor_limit,         /* variable */
        COLUMNBREAK,                /* column break */
        tag,                        /* variable */
        units,                      /* variable */
        damping,                    /* variable */
        transfer_function            /* variable */
    }
}

```

Figure 26 – Exemple d'EDD qui utilise un COLUMNBREAK

Un COLUMNBREAK dans la Figure 26 est utilisé pour indiquer un saut de colonne. Tous les éléments qui précèdent le COLUMNBREAK seront placés dans une colonne; tous les éléments ci-dessus seront rangés dans la colonne suivante (voir Figure 27).

Figure 27 – Exemple d'application EDD pour une fenêtre de présentation

Un ROWBREAK dans la Figure 28 est utilisé pour placer les éléments du menu suivants en commençant par le côté gauche (voir Figure 29).

```

MENU overview_page
{
    LABEL "Overview";
    STYLE PAGE;
    ITEMS
    {
        tag, /* variable */
        ROWBREAK,
        range_values, /* group */
        COLUMNBREAK,
        sensor_limits, /* group */
        ROWBREAK,
        units, /* variable */
        damping, /* variable */
        transfer_function /* variable */
        COLUMNBREAK
    }
}

MENU range_values
{
    LABEL "Range Values";
    STYLE GROUP;
    ITEMS
    {
        upper_range_value, /* variable */
        lower_range_value, /* variable */
    }
}

MENU sensor_limits
{
    LABEL "Sensor Limits";
    STYLE GROUP;
    ITEMS
    {
        upper_sensor_limit, /* variable */
        lower_sensor_limit, /* variable */
    }
}
    
```

Figure 28 – Exemple d'EDD pour une fenêtre de présentation

La Figure 29 montre le résultat de la Figure 28 dans une application EDD. Les encadrés en pointillés bleus montrent l'espace disponible.

Overview

TAG: PIC1023

Range Values

Upper: 60 mBar

Lower: 20 mBar

Sensor Limits

Upper: 100 mBar

Lower: 0 mBar

Unit: mBar

Damping: 20 sec

Transfer Function: Linear

Figure 29 – Exemple d'application EDD pour une fenêtre de présentation

4.5.4.3 Graphiques et diagrammes en ligne

Les graphiques et diagrammes peuvent apparaître dans les colonnes, tout comme les variables et les méthodes. Voir la Figure 30 et la Figure 31.

```
MENU overview_page
{
  LABEL "Overview";
  STYLE PAGE;
  ITEMS
  {
    primary_variable, /* variable */
    upper_range_value, /* variable */
    lower_range_value, /* variable */
    COLUMNBREAK, /* column break */
    pv_graph, /* graph */
  }
}
```

Figure 30 – Exemple d'EDD pour les graphiques et diagrammes en ligne

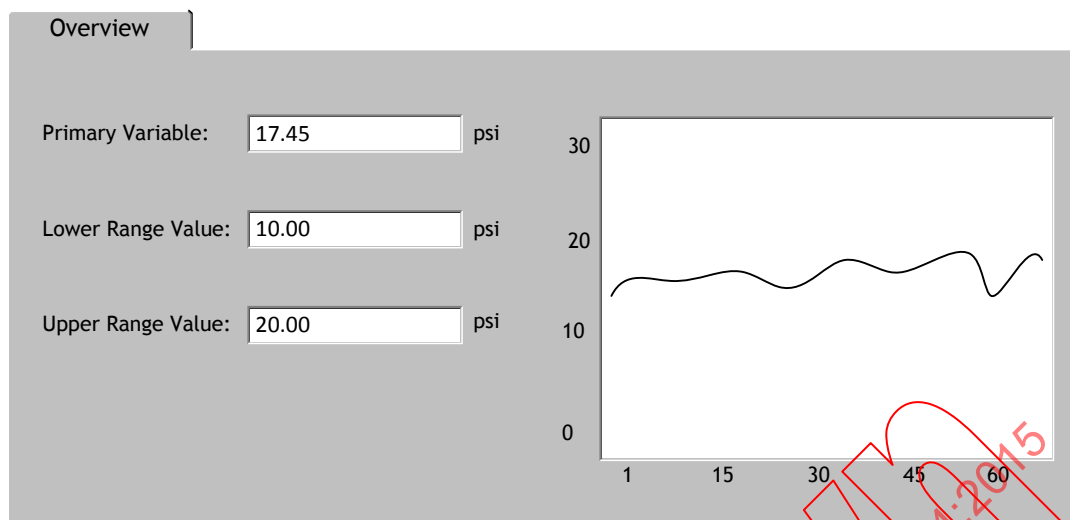


Figure 31 – Exemple d'application EDD pour un graphique en ligne

Si un graphique ou un diagramme possède un attribut WIDTH équivalent à XX_SMALL, X_SMALL, SMALL ou MEDIUM, il doit être affiché dans la colonne. Lorsque la valeur WIDTH équivaut à LARGE, X_LARGE ou XX_LARGE, la description en 4.5.4.4 doit s'appliquer.

4.5.4.4 Graphiques et diagrammes pleine largeur

Les graphiques et diagrammes peuvent également couvrir plusieurs colonnes. Voir la Figure 32 et la Figure 33.

```
GRAPH pv_graph
```

```
{
    WIDTH      MEDIUM;
    HEIGHT     MEDIUM;
    ...
}

MENU overview_page
{
    LABEL "Overview";
    STYLE PAGE;
    ITEMS
    {
        primary_variable,      /* variable */
        COLUMNBREAK,          /* column break */
        upper_range_value,     /* variable */
        COLUMNBREAK,          /* column break */
        lower_range_value,     /* variable */
        pv_graph,              /* graph */
        reload_pv_graph,       /* method */
        COLUMNBREAK,          /* column break */
        save_pv_graph          /* method */
    }
}
```

Figure 32 – Exemple d'EDD pour les graphiques et diagrammes pleine largeur

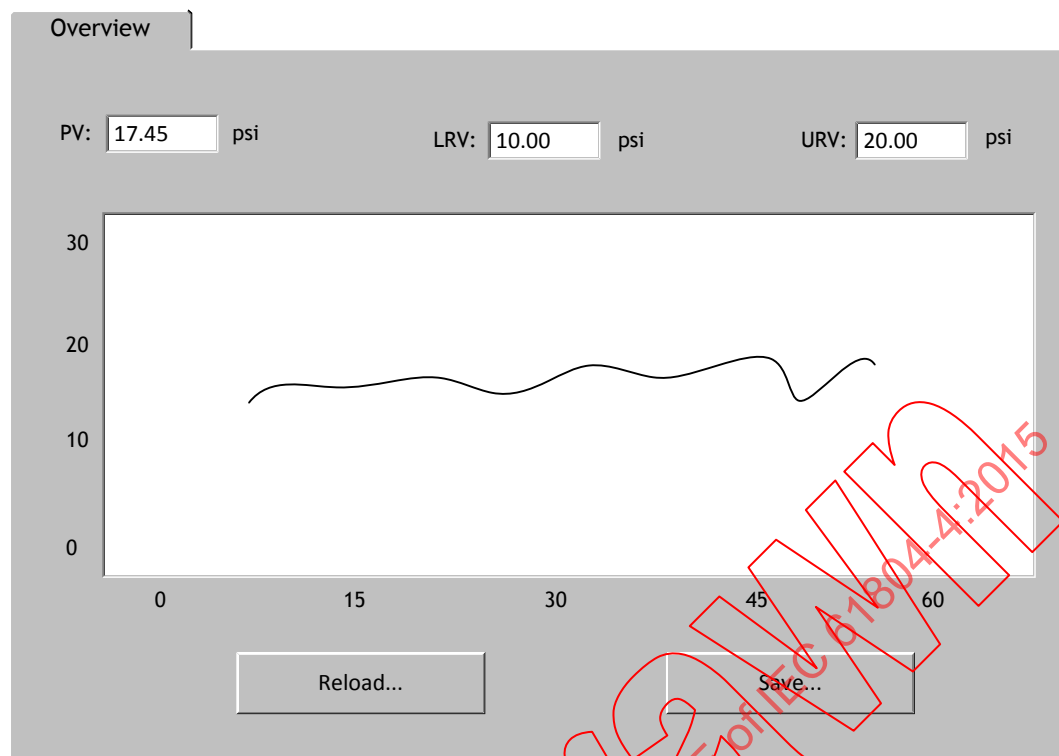


Figure 33 – Exemple d'application EDD pour un graphique pleine largeur

Si un graphique ou un diagramme possède un attribut WIDTH équivalent à LARGE, X_LARGE ou XX_LARGE, il va couvrir la largeur de la fenêtre, de la boîte de dialogue, de la page ou du groupe. Il peut exister des éléments supplémentaires avant et après le graphique. Dans ce cas, les éléments précédant le graphique ou diagramme sont séparés en un groupe de colonnes dans une ligne séparée et les éléments suivant le graphique ou diagramme sont divisés en un autre groupe de colonnes dans une ligne séparée.

NOTE Il y a trois colonnes avant le graphique et deux colonnes après celui-ci.

4.5.4.5 Conteneurs imbriqués

Un conteneur peut être imbriqué dans d'autres conteneurs. La Figure 34 et la Figure 35 représentent une page imbriquée dans une fenêtre et deux groupes imbriqués dans la page.

```

MENU overview_window
{
    LABEL "Device Overview";
    STYLE WINDOW;
    ITEMS
    {
        overview_page                                /* page */
    }
}
MENU overview_page
{
    LABEL "Overview";
    STYLE PAGE;
    ITEMS
    {
        range_values,                                /* group */
        sensor_limits,                               /* group */
        COLUMNBREAK,                                 /* column break */
        tag,                                           /* variable */
        units,                                         /* variable */
        damping,                                       /* variable */
        transfer_function                             /* variable */
    }
}
MENU range_values
{
    LABEL "Range Values";
    STYLE GROUP;
    ITEMS
    {
        upper_range_value,                           /* variable */
        lower_range_value                             /* variable */
    }
}
MENU sensor_limits
{
    LABEL "Sensor Limits";
    STYLE GROUP;
    ITEMS
    {
        upper_sensor_limit,                           /* variable */
        lower_sensor_limit                             /* variable */
    }
}

```

Figure 34 – Exemple d'EDD pour les conteneurs imbriqués

The screenshot shows a graphical user interface titled "Device Overview". It features a nested container structure. The main container has a tab labeled "Overview". Inside this container, there are two sub-containers: "Range Values" and "Sensor Limits". The "Range Values" container contains two input fields: "Upper Range Value" with a value of 20 and "Lower Range Value" with a value of 10, both followed by a "psi" unit label. The "Sensor Limits" container contains two input fields: "Upper Sensor Limit" with a value of 100 and "Lower Sensor Limit" with a value of 0, both followed by a "psi" unit label. To the right of these sub-containers, there are four more input fields: "Tag" with the value "PT-101", "Units" with a dropdown menu showing "psi", "Damping" with a value of 15 and a "sec" unit label, and "Transfer Function" with a dropdown menu showing "Linear".

Figure 35 – Exemple d'application EDD pour les conteneurs imbriqués

4.5.4.6 Edition des affichages

Les EDIT_DISPLAYS peuvent être référencées dans des conteneurs. Lorsqu'une édition d'affichage apparaît dans un conteneur, elle doit être représentée sous forme de bouton ou de lien hypertexte. Lorsque le bouton est activé, une boîte de dialogue qui affiche le contenu de l'édition d'affichage doit apparaître (voir Figure 36 et Figure 37). Les boutons OK et Annuler ne font pas partie du domaine d'application de cette spécification.

```

MENU overview_window
{
    LABEL "Device Overview";
    STYLE WINDOW;
    ITEMS
    {
        overview_page          /* page */
    }
}
MENU overview_page
{
    LABEL "Overview";
    STYLE PAGE;
    ITEMS
    {
        tag,                    /* variable */
        units,                  /* variable */
        damping,                /* variable */
        transfer_function,      /* variable */
        range_values            /* edit display */
    }
}
EDIT_DISPLAY range_values
{
    LABEL "Range Values";
    DISPLAY_ITEMS
    {
        upper_sensor_limit,    /* variable */
        lower_sensor_limit    /* variable */
    }
    EDIT_ITEMS
    {
        upper_range_value,     /* variable */
        lower_range_value,     /* variable */
        units                  /* variable */
    }
}

```

Figure 36 – Exemple d'EDD pour les éléments EDIT_DISPLAY

The image shows two overlapping windows from a software interface. The background window is titled 'Device Overview' and contains the following fields: 'Tag' with the value 'PT-101', 'Units' with 'psi', 'Damping' with '15', and 'Transfer Function' with 'Linear'. There is a 'Range Values...' button at the bottom. The foreground window is titled 'Range Values' and contains: 'Upper Sensor Limit' (100 psi), 'Lower Sensor Limit' (0 psi), 'Upper Range Value' (20 psi), 'Lower Range Value' (10 psi), and 'Units' (psi). It also has 'OK' and 'Cancel' buttons.

Figure 37 – Exemple d'application EDD pour les éléments EDIT_DISPLAY

4.5.4.7 Images

Les images peuvent également être placées dans des conteneurs. Si une image est référencée par un élément de menu INLINE, l'image s'affiche avec la colonne courante du conteneur (voir Figure 38 et Figure 39). Sinon, l'image s'affiche sur la largeur du conteneur.

```
MENU overview_page
{
  LABEL "Overview";
  STYLE PAGE;
  ITEMS
  {
    voltage,           /* Reference to an image */
    ramp_start,        /* variable */
    ramp_end,          /* variable */
    COLUMNBREAK,
    logo(INLINE)       /* Reference to an image which is displayed inline */
  }
}
```

Figure 38 – Exemple d'EDD pour les images

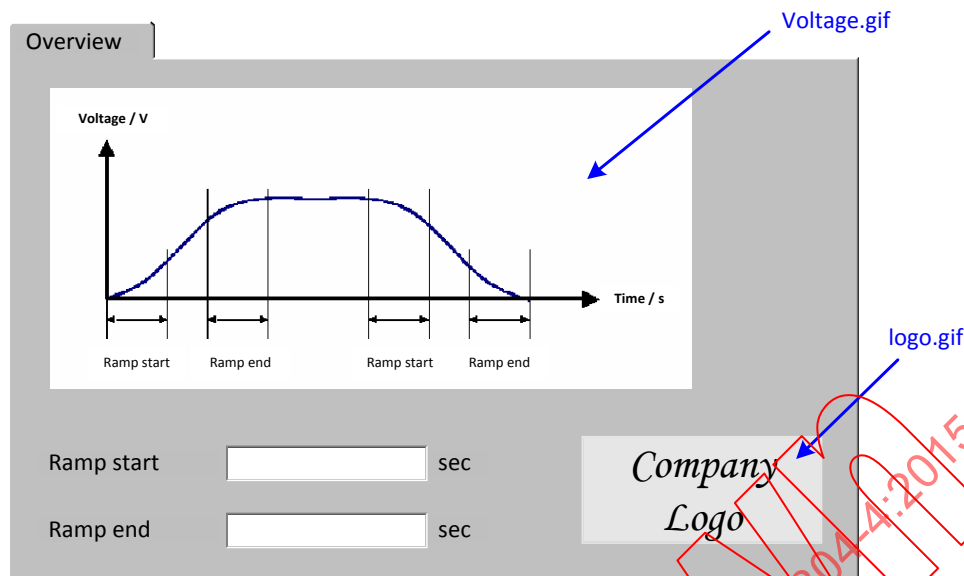


Figure 39 – Exemple d'application EDD pour les images

4.5.5 Interface utilisateur conditionnelle

4.5.5.1 Vue d'ensemble

Pour contrôler l'apparence des éléments d'interface utilisateur, l'EDDL peut procéder de plusieurs façons:

- l'attribut VISIBILITY;
- les expressions conditionnelles dans MENU ITEM;
- l'attribut VALIDITY.

4.5.5.2 VISIBILITY

L'attribut VISIBILITY des constructions de base (par exemple, MENU, VARIABLE, IMAGE, GRAPH, CHART, GRID) permet de contrôler l'apparence des éléments d'interface utilisateur. Les valeurs TRUE ou FALSE de l'attribut VISIBILITY sont destinées à être évaluées de manière dynamique en fonction des valeurs VARIABLE.

Dans le cas d'un MENU, l'attribut VISIBILITY du MENU enfant n'affecte pas la disposition d'écran du MENU. Dans la disposition d'écran du MENU, le même espace est réservé, comme si le MENU enfant était affiché.

4.5.5.3 Expressions conditionnelles dans MENU ITEM

Les expressions conditionnelles de la liste MENU ITEM permettent de contrôler la disposition d'écran. Dans la disposition d'écran, aucun espace n'est réservé si l'élément n'est pas affiché.

4.5.5.4 VALIDITY

L'attribut VALIDITY des constructions de base, par exemple MENU, VARIABLE, IMAGE, GRAPH, CHART, GRID, permet de contrôler l'apparence des éléments d'interface utilisateur. Les valeurs TRUE ou FALSE de l'attribut VALIDITY sont destinées à être évaluées de manière dynamique en fonction des valeurs VARIABLE.

VALIDITY affecte la disposition d'écran. Dans la disposition d'écran, aucun espace n'est réservé si l'élément n'est pas valide.

NOTE La communication est influencée par l'attribut VALIDITY. Voir l'Article 10.

4.6 Eléments graphiques

4.6.1 Vue d'ensemble

Les appareils de terrain équipés d'un microprocesseur intelligent deviennent de plus en plus sophistiqués et complexes. Dans certains cas, les mesures ou contrôles qui étaient auparavant irréalisables ou qui exigeaient de nombreux équipements ont été intégrés en un seul appareil. EDDL prend en charge ces appareils très sophistiqués.

Ces constructions peuvent évidemment être utilisées dans de nombreux autres types d'appareils. Le développeur EDD a le contrôle total sur le contenu de l'EDD. Il lui incombe de décider quelles constructions EDDL doivent être utilisées.

Afin de répondre aux besoins de ces appareils complexes, la prise en charge des capacités suivantes a été ajoutée:

- élaboration de graphiques;
- élaboration de diagrammes;
- gestion du contenu des images;
- gestion des données tabulaires.

EDDL prend en charge deux types de représentations des données graphiques. Il s'agit des représentations GRAPH et CHART. Un CHART est utilisé pour afficher les valeurs réelles et un GRAPH sert à afficher les données stockées qui peuvent être lues à partir de l'appareil ou d'une mémoire permanente. Il est fréquent de comparer une forme d'onde de l'appareil à une forme d'onde de référence ou d'une date/opération spécifique stockée par l'application EDD. La principale différence entre ces deux constructions est la suivante: l'application EDD gère les données d'un CHART et l'EDD définit, lit et met à jour lui-même les données. Les méthodes de gestion des données sont facultatives pour les CHART, mais généralement nécessaires pour les GRAPH.

Un GRAPH contient une liste de WAVEFORMs tracées conjointement. Le GRAPH peut être référencé depuis un MENU ou dans le cadre d'une METHOD. Les WAVEFORMs ont un minimum d'attributs obligatoires pour permettre au développeur EDD de produire facilement un graphique. Pour les développeurs souhaitant davantage de contrôle, une large gamme d'attributs facultatifs est disponible (taille d'affichage, axe, points clés, etc.).

Ces constructions sont particulièrement utiles pour les signatures de vanne de tracé et les signaux radar. Les données de l'appareil de terrain et celles stockées par l'application EDDL peuvent être tracées sur le même GRAPH. Par exemple, les données de l'appareil de terrain peuvent être spécifiées dans une WAVEFORM et les données permanentes d'un FILE dans une autre WAVEFORM. Les deux WAVEFORM peuvent être tracées sur le même GRAPH.

Un GRAPH est un instantané de l'appareil ou des propriétés du processus alors qu'un CHART apporte une vue d'une tendance évoluant avec le temps. Un CHART possède des SOURCES qui sont prélevées régulièrement. Les tracés de style STRIP, SWEEP et SCOPE sont pris en charge afin de montrer les tendances au fil du temps. Les types HORIZONTAL, VERTICAL et GAUGE permettent une représentation graphique d'une seule valeur instantanée.

L'état et la performance de certains appareils ne peuvent pas être déterminés empiriquement. Pour ces appareils en question, la performance relative est l'indicateur de l'état de l'appareil. La construction FILE permet d'accéder aux données stockées de l'appareil par l'application EDDL. L'EDD précise les données, qui doivent être stockées avec une syntaxe semblable à celle de la COLLECTION. Le développeur EDD détermine les données à stocker dans les combinaisons de VARIABLES, d'ARRAYs, de COLLECTIONs ou de LISTs.

La construction LIST a été ajoutée afin d'améliorer la flexibilité du langage lors de la spécification du stockage des données permanentes dans un FILE. La construction LIST est une matrice de longueur variable. Certaines données peuvent être ajoutées ou supprimées de la liste au fil du temps. Chaque entrée de la LIST appartient au même type que celui spécifié par le développeur EDD.

La construction COLLECTION prend en charge toutes les combinaisons des types de membres. Auparavant, les collections étaient seulement admises à contenir des références d'un seul type (VARIABLE, ARRAY, COLLECTION, MENU).

Le Paragraphe 4.6 fournit les lignes directrices et les prévisions d'utilisation et de prise en charge des constructions EDDL pour les représentations graphiques.

4.6.2 Graphique et diagramme

4.6.3 Attributs communs

4.6.3.1 Tailles

Les attributs HEIGHT et WIDTH définissent la taille du CHART et du GRAPH en fonction de la fenêtre restituée par l'application EDD. Les attributs HEIGHT et WIDTH ne fournissent aucune valeur absolue pour la fenêtre du graphique, mais plutôt une gamme de valeurs allant de XX_SMALL à XX_LARGE. L'application EDD est libre de déterminer la taille de la fenêtre du graphique actuelle. Il convient que l'application EDD rende les CHARTs et les GRAPHs qui font référence à la même valeur HEIGHT ou WIDTH dans la même proportion. Le paramètre par défaut pour la HEIGHT et la WIDTH est MEDIUM (voir Figure 40). Il convient que l'application EDD rende les CHART et les GRAPH qui possèdent la même HEIGHT ou WIDTH dans la même proportion.

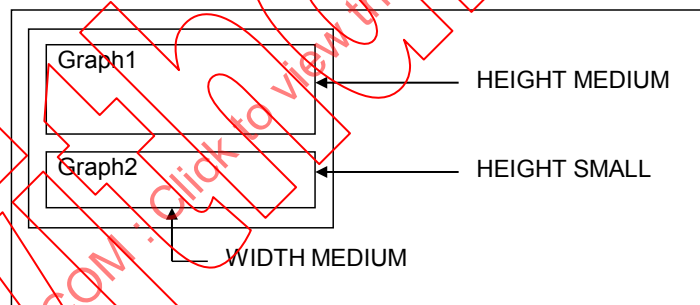


Figure 40 – Attributs HEIGHT et WIDTH pour les CHART et les GRAPH

4.6.3.2 Caractéristiques des traits

L'attribut LINE_TYPE définit les catégories d'attributs d'affichage pour les WAVEFORMs restituées sur les GRAPHs et les SOURCEs restituées sur les CHARTs. De tels attributs d'affichage peuvent inclure des styles de couleur et de traits (tirets, pointillés, etc.). L'application EDD peut fournir des attributs configurables par les utilisateurs pour chaque LINE_TYPE. Il convient de restituer les WAVEFORMs et SOURCEs qui possèdent le même attribut LINE_TYPE avec les caractéristiques. Il existe des catégories de données de DATA 1 à DATA 9, avec certaines limites et TRANSPARENT. Ainsi, seul l'affichage avec les valeurs est fourni, non le trait de connexion.

L'attribut EMPHASIS est utilisé pour différencier une ou plusieurs SOURCEs ou WAVEFORM dans un CHART ou un GRAPH donné. Par défaut, il convient d'afficher la WAVEFORM ou SOURCE avec l'attribut EMPHASIS paramétré sur TRUE à un poids plus élevé (voir Figure 41). L'application EDD peut fournir des attributs configurables par les utilisateurs pour l'attribut EMPHASIS.

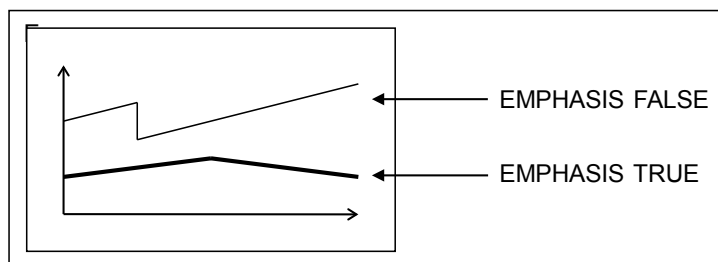


Figure 41 – Attribut EMPHASIS pour différencier une ou plusieurs SOURCES ou WAVEFORMS

4.6.4 CHART

4.6.4.1 Vue d'ensemble

Un CHART est utilisé pour afficher des valeurs de données continues. Le diagramme prend en charge de nombreuses sources de données et de nombreux axes Y. Le diagramme prend en charge de nombreuses sources de données et de nombreux axes Y. Des méthodes peuvent être définies pour la récupération de données et la prise en charge des fonctions d'agrandissement et de défilement.

La couleur de fond d'affichage d'un CHART est déterminée par l'application EDD. Afin de respecter une certaine neutralité, il est recommandé de définir le blanc comme couleur de fond d'écran. Il est recommandé qu'une application EDD prenne en charge au moins six courbes à afficher simultanément.

Les éléments d'un CHART sont la SOURCE (qui définit la source des données), les éléments de visualisation, le type et les actions du diagramme.

4.6.4.2 Types de diagrammes

4.6.4.2.1 Vue d'ensemble

Un diagramme peut être affiché de différentes manières. Il peut être affiché sous forme de diagramme à barres horizontal ou vertical, diagramme avec des formes d'ondes constamment mises à jour ou de jauge. Si le type n'est pas défini, la valeur par défaut est STRIP.

4.6.4.2.2 GAUGE

L'élément visuel réel d'une jauge est déterminé par l'application EDD. Un CHART de type GAUGE ne peut posséder qu'une seule source de données.

4.6.4.2.3 HORIZONTAL_BAR

Le format réel (éléments visuels) d'affichage est déterminé par l'application EDD. Ce type impose le graphique à barres horizontales comme type d'affichage. Ce type d'affichage peut contenir de nombreux diagrammes à barres (SOURCE). Un diagramme à barres est affiché pour chacune des variables.

4.6.4.2.4 SCOPE

Dans SCOPE, lorsque les valeurs sources atteignent la fin de la zone d'affichage du CHART, la zone d'affichage est effacée. Les nouvelles valeurs sources sont une nouvelle fois affichées, en commençant au début de la zone d'affichage. Ce type d'affichage peut contenir de nombreuses définitions de SOURCE.

4.6.4.2.5 STRIP

Dans STRIP, lorsque les valeurs sources atteignent la fin de la zone d'affichage du CHART, la zone d'affichage est décalée. Les valeurs sources les plus anciennes sont alors effacées de la zone d'affichage et les nouvelles valeurs sources sont ajoutées à leur place. Ce type d'affichage peut contenir de nombreuses définitions de SOURCE.

4.6.4.2.6 SWEEP

Dans SWEEP, lorsque les valeurs sources atteignent la fin de la zone d'affichage du CHART, les nouvelles valeurs sources sont à nouveau affichées à partir du début de la zone d'affichage. Contrairement à SCOPE, seule la partie de la zone d'affichage nécessaire à l'affichage des nouvelles valeurs sources est effacée. Ce type d'affichage peut contenir de nombreuses définitions de SOURCE.

4.6.4.2.7 VERTICAL_BAR

Le format réel (éléments visuels) d'affichage est déterminé par l'application EDD. Ce type impose le graphique à barres verticales comme type d'affichage. Ce type d'affichage peut contenir de nombreux diagrammes à barres. Un diagramme à barres est affiché pour chacune des variables.

4.6.4.3 Application EDDL sans prise en charge totale du diagramme

Il convient que les applications EDDL qui ne prennent pas en charge la vue graphique du diagramme affichent les variables associées de manière numérique normalisée. Par exemple, les données peuvent être affichées sous forme de tableaux. Il convient que le format soit choisi par le développeur de l'application EDD.

4.6.4.4 Longueur et durée de cycle

L'intervalle de temps affiché sur l'axe du temps est défini par l'attribut LENGTH. La durée de cycle définit l'intervalle entre les relevés de variables de l'application EDD (en ms). L'application EDD met à jour l'axe du temps avec le temps des données visibles. Si l'application ne peut pas lire aussi rapidement que la durée de cycle le définit, elle lit simplement les données aussi rapidement qu'elle le peut.

4.6.4.5 Sources de données d'un diagramme

Un diagramme peut afficher une ou plusieurs courbes. Pour chacune des courbes, une variable est utilisée. Pour prendre ces actions en charge, le CHART fait référence à une ou plusieurs sources. La SOURCE peut faire référence à une ou plusieurs variables.

La Figure 42 montre un diagramme avec une courbe dans une boîte de dialogue. La courbe est réactualisée dans une durée de cycle de 1 s.

```

MENU measuring_values
{
    LABEL "Measuring Values";
    STYLE DIALOG;
    ITEMS
    {
        primary_value_view
    }
}

MENU primary_value_view
{
    LABEL "Primary Measuring Value";
    STYLE PAGE;
    ITEMS
    {
        primary_value_chart
    }
}

VARIABLE primary_value
{
    LABEL "Primary Value";
    CLASS DYNAMIC;
    TYPE FLOAT;
    CONSTANT_UNIT "Bar";
}

SOURCE primary_value_source
{
    MEMBERS
    {
        PRIM_VAL, primary_value;
    }
}

CHART primary_value_chart
{
    LABEL "Primary Value";
    MEMBERS
    {
        CHART1, primary_value_source;
    }
}

```

Figure 42 – Exemple d'un diagramme avec une courbe dans une boîte de dialogue

4.6.4.6 Les CHARTs avec plusieurs SOURCES faisant référence au même AXIS

Une application EDD doit associer des SOURCES faisant référence au même AXIS. En cas de CHARTs de TYPE HORIZONTAL_BAR et VERTICAL_BAR, les différentes barres doivent être affichées ensemble avec un axe. En cas de CHART de TYPE GAUGE, SCOPE, STRIP et SWEEP, les courbes doivent être affichées ensemble dans la zone du diagramme. Un seul axe doit être montré sur un côté de la zone du diagramme.

La Figure 43 est un exemple d'EDD qui montre l'association de deux SOURCES. La Figure 44 montre comment une application EDD peut afficher le diagramme.

```

VARIABLE primary_value
{
    LABEL "Primary Value";
    CLASS DYNAMIC;
    TYPE FLOAT;
    CONSTANT_UNIT "pH";
}

VARIABLE primary_value_undamped
{
    LABEL "Undamped PV";
    CLASS DYNAMIC;
    TYPE FLOAT;
    CONSTANT_UNIT "pH";
}

AXIS axis_0_14
{
    LABEL "axis 0-14";
    MIN_VALUE 0;
    MAX_VALUE 14;
}

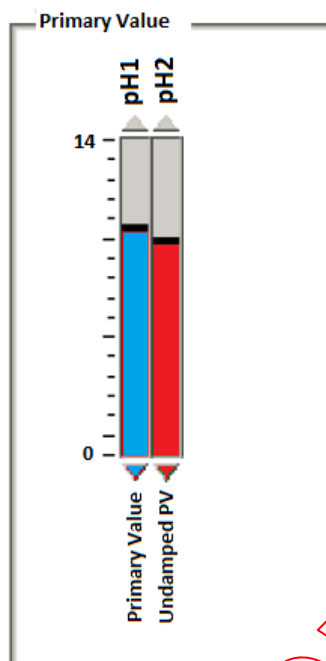
SOURCE primary_value_stack1
{
    MEMBERS
    {
        PRIM_VAL_1, primary_value;
    }
    Y_AXIS axis_0_14;
    LINE_COLOR 0x0000FF; /*BLUE*/
}

SOURCE primary_value_stack2
{
    MEMBERS
    {
        PRIM_VAL_2, primary_value_undamped;
    }
    Y_AXIS axis_0_14;
    LINE_COLOR 0xFF0000; /*RED*/
}

CHART primary_value_stack_chart
{
    LABEL "Primary Value";
    MEMBERS
    {
        CHART1, primary_value_stack1, "ph1";
        CHART2, primary_value_stack2, "pH2";
    }
    TYPE VERTICAL_BAR;
}

```

Figure 43 – Exemple de diagramme avec deux SOURCES



Anglais	Français
Primary Value	Valeur principale
Undamped PV	PV sans amortissement

Figure 44 – Affichage d'un exemple de diagramme avec deux SOURCES

La Figure 45 est un exemple d'EDD qui présente l'association de trois SOURCES horizontales. La Figure 46 montre comment une application EDD peut l'afficher.

```

VARIABLE primary_value
{
    LABEL "Primary Value";
    CLASS DYNAMIC;
    TYPE FLOAT;
    CONSTANT_UNIT "pH";
}

VARIABLE primary_value_undamped
{
    LABEL "Undamped PV";
    CLASS DYNAMIC;
    TYPE FLOAT;
    CONSTANT_UNIT "pH";
}

VARIABLE primary_value_non_temperature_compensated
{
    LABEL "PV non-TC";
    CLASS DYNAMIC;
    TYPE FLOAT;
    CONSTANT_UNIT "pH";
}

AXIS axis_0_14
{
    LABEL "axis 0-14";
    MIN_VALUE 0;
    MAX_VALUE 14;
}

AXIS axis_0_7
{
    LABEL "axis 0-7";
    MIN_VALUE 0;
    MAX_VALUE 7;
}

SOURCE primary_value_stack1
{
    MEMBERS
    {
        PRIM_VAL_1, primary_value;
    }
    Y_AXIS axis_0_14;
    LINE_COLOR 0x0000FF; /*BLUE*/
}

SOURCE primary_value_stack2
{
    MEMBERS
    {
        PRIM_VAL_2, primary_value_undamped;
    }
    Y_AXIS axis_0_14;
    LINE_COLOR 0xFF0000; /*RED*/
}

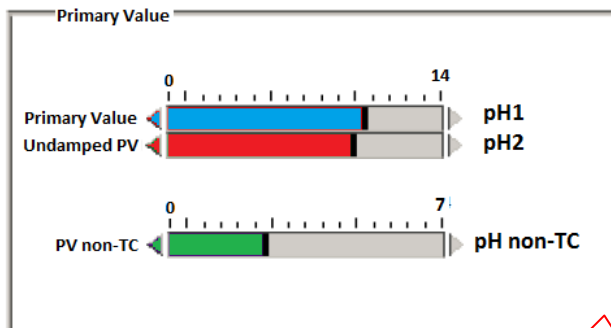
SOURCE primary_value_single
{
    MEMBERS
    {
        PRIM_VAL_3, primary_value_non_temperature_compensated;
    }
    Y_AXIS axis_0_7;
    LINE_COLOR 0x008000; /*GREEN*/
}

CHART primary_value_stack_chart
{
    LABEL "Primary Value";
    MEMBERS
    {
        CHART1, primary_value_stack1, "pH1";
        CHART2, primary_value_single, "pH non-TC";
        CHART3, primary_value_stack2, "pH2";
    }
    TYPE HORIZONTAL_BAR;
}

```

)

Figure 45 – Exemple de diagramme avec trois barres horizontales



Anglais	Français
Primary value	Valeur principale
Undamped PV	PV sans amortissement
PV non-TC	PV non TC
pH non-TC	pH non TC

Figure 46 – Affichage d'un exemple de diagramme avec trois barres horizontales

4.6.4.7 Légende et aide pour les courbes

L'EDD peut fournir des informations sur les libellés et les aides concernant les CHART et leurs courbes et axes. Afin de créer des légendes et des informations d'aide pour les courbes, l'application EDD doit prendre en charge les règles ci-après.

L'application EDD doit afficher la description si elle est définie comme une légende par les membres du CHART. L'application EDD doit utiliser le LABEL de la SOURCE référencée si la description n'est pas définie. Une chaîne vide constitue une chaîne définie.

Quand les SOURCES ont plusieurs variables, l'application EDD doit afficher la description si elle est définie comme une légende par les membres de la SOURCE. L'application EDD doit utiliser le LABEL de la VARIABLE référencée si la description n'est pas définie.

L'application EDD doit fournir les informations d'aide des membres du CHART pour les utilisateurs.

Si l'attribut HELP n'existe pas dans les membres du CHART, les informations d'aide des SOURCES doivent être utilisées. Si l'attribut HELP n'existe pas dans les SOURCES, les informations d'aide des membres de la SOURCE doivent être utilisées. Si l'attribut HELP n'existe pas dans le CHART, les SOURCES et les membres de la SOURCE, les informations d'aide des VARIABLES doivent être utilisées.

4.6.4.8 Fonctions d'agrandissement et de défilement

L'application EDD peut prendre en charge les fonctions d'agrandissement et de défilement. L'EDD ne nécessite donc pas de prise en charge. L'application EDD lit les variables d'un diagramme et les enregistre dans son propre stockage. Lorsqu'elle utilise les fonctions d'agrandissement et de défilement, l'application EDD montre simplement moins, plus ou d'autres points des données de stockage dans la zone d'affichage.

4.6.4.9 Actions

Les INIT_ACTIONS et/ou les REFRESH_ACTIONS peuvent être définies dans une SOURCE. Il convient de définir les variables qui sont référencées dans la SOURCE dans les méthodes. La valeur peut être calculée à l'aide des variables d'appareil et/ou des variables locales.

Si des INIT_ACTIONS sont définies, l'application EDD appelle ces méthodes au lieu de lire les variables.

Si des REFRESH_ACTIONS sont définies, l'application EDD appelle ces méthodes de manière cyclique selon l'intervalle CYCLE_TIME au lieu de lire les variables.

La même méthode peut être référencée dans les listes de méthodes des INIT_ACTIONS et des REFRESH_ACTIONS.

La Figure 47 est un exemple d'utilisation des méthodes dans un diagramme. Cet exemple représente un diagramme dans une boîte de dialogue.

```

MENU measuring_values
{
    LABEL "Measuring Values";
    STYLE DIALOG;
    ITEMS
    {
        primary_value_view
    }
}

MENU primary_value_view
{
    LABEL "Primary Measuring Value";
    STYLE PAGE;
    ITEMS
    {
        primary_value_chart
    }
}

VARIABLE primary_value
{
    LABEL "Primary Value";
    CLASS DYNAMIC;
    TYPE FLOAT;
}

SOURCE primary_value_source
{
    LABEL "Primary";                                     // this label is used in the legend
    LINE_TYPE DATA1;
    EMPHASIS TRUE;
    Y_AXIS measuring_values_axis;
    MEMBERS
    {
        PRIM_VAL, primary_value;
    }
}

METHOD CalculationMethod
{
    LABEL "";
    DEFINITION
    {
        calculated_value = primary_value * 0.12345;
    }
}

AXIS measuring_values_axis
{
    LABEL "measurement value";
    MIN_VALUE 0;
    MAX_VALUE 100;
}

```

```

VARIABLE primary_value_unit
{
    LABEL "Primary Value Unit";
    CLASS CONTAINED;
    TYPE ENUMERATED(1)
    {
        { 32, [degC], [degC_help] },
        { 33, [degF], [degF_help] },
        { 35, [Kelvin], [Kelvin_help] }
    }
}

UNIT
{
    primary_value_unit:
    primary_value,
    calculated_value,
    measuring_values_axis
}

SOURCE calculated_value_source
{
    LABEL "calculated"; // this label is used in the legend
    LINE_TYPE DATA2;
    Y_AXIS measuring_values_axis;
    MEMBERS
    {
        CALC_VAL, calculated_value;
    }
    INIT_ACTIONS {CalculationMethod}
    REFRESH_ACTIONS {CalculationMethod}
}

CHART primary_value_chart
{
    LABEL "Primary Value";
    LENGTH 600000;
    TYPE SCOPE;
    WIDTH LARGE;
    HEIGHT SMALL;
    MEMBERS
    {
        GRAPH1, primary_value_source,
        GRAPH2, calculated_value_source;
    }
}
    
```

Figure 47 – Exemple d'un diagramme dans une boîte de dialogue

4.6.5 GRAPH

4.6.5.1 Vue d'ensemble

Une solution GRAPH évolutive est prise en charge par les applications EDDL avec des formes d'onde et des définitions d'axe. On peut définir plusieurs formes d'onde qui sont basées sur un ou plusieurs axes y et un axe x. Certaines méthodes peuvent être utilisées pour la récupération des données et pour les fonctions d'agrandissement et de défilement.

La couleur de fond d'affichage d'un GRAPH est déterminée par l'application EDD. Afin de respecter une certaine neutralité, il est recommandé de définir le blanc comme couleur de fond. Il convient qu'une application EDD prenne en charge l'affichage simultané d'au moins six courbes.

Un graphique est constitué de deux éléments principaux: la WAVEFORM et l'AXIS. La WAVEFORM fournit la source et le type de données, l'accentuation à appliquer, les éléments visuels et toutes les actions à effectuer lors de l'initialisation ou la réactualisation. Un seul graphique peut comporter plusieurs WAVEFORM. Les données comme les traits affichant les limites et les marqueurs d'un graphique pourraient être incluses. Aucun plafond n'est défini pour le nombre de WAVEFORM dans un GRAPH. Les WAVEFORM contiennent également une référence à une définition Y_AXIS. Lors de l'utilisation de plusieurs WAVEFORM, il

convient de faire référence au même Y_AXIS. Si les WAVEFORM utilisent différents axes, chacun peut avoir une mise à l'échelle et une unité différentes.

La Figure 48 capture un graphique et les éléments visuels qui sont fournis par les constructions définies en EDDL. Chacun de ces éléments sera affiché de manière plus détaillée dans les articles en lien.

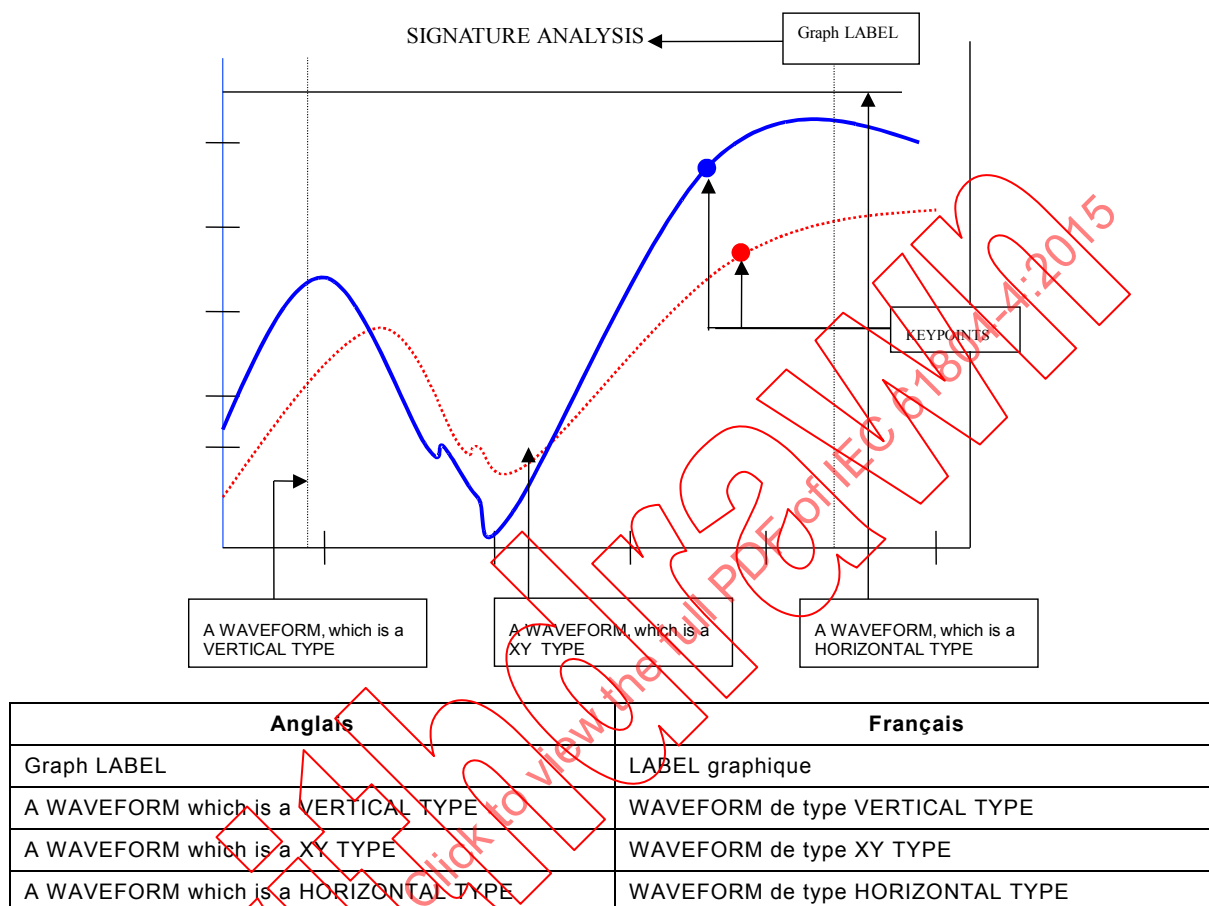


Figure 48 – Le graphique et les éléments visuels

4.6.5.2 Types

La construction GRAPH possède des attributs WAVEFORM qui définissent les données qui peuvent être affichées sur le GRAPH. La WAVEFORM peut être utilisée pour tracer les valeurs ou enveloppes limites dans un affichage. Pour cela, les types HORIZONTAL et VERTICAL sont utilisés. La construction WAVEFORM fournit les attributs TYPE qui choisissent entre XY, YT, HORIZONTAL ou VERTICAL.

Le type XY fournit un ensemble des deux listes des points X et Y. Il convient que le développeur EDD s'assure que la position de la "valeur x" dans la liste X soit la même que la position de la "valeur y" correspondante dans la liste Y. Le nombre de points est également défini dans la construction. Si aucun nombre n'est défini, l'application EDD la configure au nombre de points dans la liste. Cela s'avérerait utile dans les situations pour lesquelles le nombre de points n'est pas connu.

Une WAVEFORM de type YT fournit un ensemble de valeurs Y. La valeur X initiale et les incréments sont définis pour générer les valeurs X ultérieures. On utiliserait normalement ce cas pour représenter les formes d'onde qui sont échantillonnées de manière périodique de sorte que X se répète à intervalles réguliers. Le nombre de points est défini dans la forme d'onde. En l'absence du nombre de points, l'application EDD utilise le nombre de valeurs Y comme nombre de points.

Une WAVEFORM de type HORIZONTAL permet à l'utilisateur de tracer des lignes horizontales sur l'écran graphique. On utiliserait normalement ce cas pour indiquer des limites minimales et maximales ou un cadre englobant. La construction comporte une série de valeurs Y, chaque valeur Y spécifiant une ligne horizontale.

Une WAVEFORM de type VERTICAL permet à l'utilisateur de tracer des lignes verticales sur l'écran graphique. On utiliserait normalement ce cas pour indiquer des limites minimales et maximales ou un cadre englobant. La construction comporte une série de valeurs X, chaque valeur X spécifiant une ligne verticale.

Les caractéristiques visuelles d'une WAVEFORM sont spécifiées par l'attribut LINE_TYPE. Une WAVEFORM est visualisée sous la forme d'une série de points lorsque l'attribut LINE_TYPE est spécifié comme TRANSPARENT.

4.6.5.3 HANDLING

Cet attribut définit le type d'opérations pouvant être réalisées sur la WAVEFORM (lecture, lecture/écriture ou écriture).

4.6.5.4 KEY_POINTS

Cet attribut définit certains points accentués sur le graphique. Les valeurs X et Y du point clé sont fournies dans la WAVEFORM. L'application EDD a besoin d'accentuer ce point sur l'affichage. La détermination du mode d'accentuation appliqué est laissée à l'appréciation de l'application EDD.

4.6.5.5 Actions

4.6.5.5.1 INIT_ACTIONS

Cet attribut définit l'ensemble d'actions devant être exécutées avant l'affichage initial de la WAVEFORM. On le spécifie sous la forme de références aux instances de METHOD qui doivent être appelées dans un ordre particulier. En cas d'échec ou d'interruption d'une METHOD pour une quelconque raison, l'affichage de la WAVEFORM est interrompu. Cette construction peut être utilisée pour la lecture initiale de la WAVEFORM à partir de l'appareil ou d'un FILE ainsi que pour le prétraitement nécessaire (par exemple, moyennage ou filtrage de données) qui est effectué avant l'affichage.

4.6.5.5.2 REFRESH_ACTIONS

L'attribut REFRESH_ACTIONS fournit des références aux méthodes devant être exécutées lorsque des modifications sont apportées à l'axe X ou Y ou lorsqu'un attribut CYCLE_TIME est défini dans le GRAPH chaque fois que le CYCLE_TIME s'écoule. Les méthodes sont exécutées dans l'ordre de définition. En cas d'échec ou d'interruption d'une méthode, les méthodes restantes ne sont pas exécutées et le GRAPH revient à l'affichage précédent.

La méthode peut lire les données dans les sections si la taille des données est tellement volumineuse qu'elles ne peuvent pas être transférées lors d'un transfert de communication.

Dans la méthode, la zone visible peut être calculée et définie sur l'axe au moyen des valeurs VIEW_MIN et VIEW_MAX. Toutes les formes d'onde associées au même axe doivent être représentées avec la même zone du graphique.

La Figure 49 est un exemple de graphique simple.

4.6.5.5.3 EXIT_ACTIONS

Les EXIT_ACTIONS sont appelées si la forme d'onde disparaît de l'écran.

Les actions de sortie permettent aux données de forme d'onde éditées d'être capturées et manipulées selon le besoin par l'EDD.

De plus, si l'appareil a besoin d'être dans un mode spécial (par exemple, diagnostic) pour fournir des données de forme d'onde, ce mode peut être redéfini à la fermeture du graphique.

La Figure 49 est un exemple de graphique.

```

VARIABLE      x_value
{
    LABEL "...";
    HELP "...";
    CLASS DYNAMIC;
    TYPE FLOAT;
    CONSTANT_UNIT [degC];
}

ARRAY          x_data
{
    LABEL "x-data";
    NUMBER_OF_ELEMENTS 1000;
    TYPE x_value;
}

VARIABLE      y_value
{
    LABEL "...";
    HELP "...";
    CLASS DYNAMIC;
    TYPE FLOAT;
    CONSTANT_UNIT [degC];
}

ARRAY          y_data
{
    LABEL "y-data";
    NUMBER_OF_ELEMENTS 1000;
    TYPE y_value;
}

VARIABLE      x_min_value
{
    LABEL "...";
    HELP "...";
    CLASS LOCAL;
    TYPE FLOAT;
}

AXIS x_axis_signature
{
    LABEL "x axis";
    MIN_VALUE DEFAULT_X_MIN_VALUE;
    MAX_VALUE DEFAULT_X_MAX_VALUE;
}

AXIS y_axis_signature
{
    LABEL "y axis";
    MIN_VALUE DEFAULT_Y_MIN_VALUE;
    MAX_VALUE DEFAULT_Y_MAX_VALUE;
}

WAVEFORM value_signature1
{
    TYPE          XY
    {
        X_VALUES      { x_data }
        Y_VALUES      { y_data }
    }
    INIT_ACTIONS { read_first_signature }
    REFRESH_ACTIONS { read_signature }
    Y_AXIS y_axis_signature;
}

x_max_value      LIKE x_min_value;

```

```

device_x_min_value LIKE x_min_value;
device_x_max_value LIKE x_min_value;

y_min_value      LIKE x_min_value;
y_max_value      LIKE x_min_value;
device_y_min_value LIKE x_min_value;
device_y_max_value LIKE x_min_value;

METHOD read_first_signature
{
    DEFINITION
    {
        // read data from the device depending predefined default
        // MIN_VALUE and MAX_VALUE of the x and y axis
        x_min_value = DEFAULT_X_MIN_VALUE;
        x_max_value = DEFAULT_X_MAX_VALUE;
        y_min_value = DEFAULT_Y_MIN_VALUE;
        y_max_value = DEFAULT_Y_MAX_VALUE;
        WriteCommand (write_data_area);
        ReadCommand (read_data);
        x_axis_signature.MIN_VALUE = device_x_min_value;
        x_axis_signature.MAX_VALUE = device_x_max_value;
        y_axis_signature.MIN_VALUE = device_y_min_value;
        y_axis_signature.MAX_VALUE = device_y_max_value;
    }
}

METHOD read_signature
{
    DEFINITION
    {
        // read data from the device depending of the current MIN_VALUE and
        MAX_VALUE
        // of the x axis
        x_min_value = x_axis_signature.VIEW_MIN;
        x_max_value = x_axis_signature.VIEW_MAX;
        y_min_value = y_axis_signature.VIEW_MIN;
        y_max_value = y_axis_signature.VIEW_MAX;
        WriteCommand (write_data_area);
        ReadCommand (read_data);
        if (device_x_min_value < x_min_value)
            x_axis_signature.VIEW_MIN = device_x_min_value;
        if (device_x_max_value > x_max_value)
            x_axis_signature.VIEW_MAX = device_x_max_value;
        if (device_y_min_value < y_min_value)
            y_axis_signature.VIEW_MIN = device_y_min_value;
        if (device_y_max_value > y_max_value)
            y_axis_signature.VIEW_MAX = device_y_max_value;
    }
}

GRAPH valve_signature
{
    MEMBERS
    {
        VAL_SIG; value_signature1;
    }
    X_AXIS x_axis_signature;
}

```

Figure 49 – Exemple de graphique

4.6.5.6 Axe

La définition Y_AXIS est fournie par la construction WAVEFORM tandis que la définition X_AXIS est fournie par la construction GRAPH elle-même.

Cet attribut fournit une référence concernant l'AXIS. Lorsqu'il est affiché, l'AXIS doit être tracé avec l'attribut WAVEFORM. Si cela n'est pas défini, l'application EDD construit l'AXIS à partir des valeurs minimale et maximale et d'autres règles internes éventuelles pouvant exister.

La construction AXIS définit la manière de construire l'axe X ou Y qui doivent s'afficher sur un GRAPH ou CHART. L'AXIS possède des attributs qui définissent ses valeurs minimale et

maximale. De plus, il existe un attribut SCALING qui définit s'il convient de mettre l'axe à l'échelle de façon LINEAR ou LOGARITHMIC. L'échelle logarithmique est en base 10.

Le développeur EDD peut définir un LABEL pouvant être affiché avec l'AXIS, ainsi qu'une chaîne qui spécifie les unités dans lesquelles les valeurs sont affichées sur le GRAPH.

La Figure 50 est un exemple d'EDD d'un axe commun pour les WAVEFORM w1 et w2, ainsi qu'un second axe pour la WAVEFORM w3.

```

AXIS    x1 { }
AXIS    y1 { }
AXIS    y2 { }

GRAPH graph1
{
    MEMBERS
    {
        W1, w1;
        W2, w2;
        W3, w3;
    }
    X_AXIS x1;
}

WAVEFORM w1
{
    Y_AXIS y1;
    TYPE XY
    {
        Y_VALUES {w1_values}
        X_INCREMENT 1
        X_INITIAL 0
    }
}

WAVEFORM w2
{
    Y_AXIS y1;
    TYPE XY
    {
        Y_VALUES {w2_values}
        X_INCREMENT 1
        X_INITIAL 0
    }
}

WAVEFORM w3
{
    Y_AXIS y2;
    TYPE XY
    {
        Y_VALUES {w3_values}
        X_INCREMENT 1
        X_INITIAL 0
    }
}

```

Figure 50 – Plusieurs axes utilisés

4.6.5.7 Application EDDL sans prise en charge graphique

Si une application EDDL ne prend pas en charge la vue graphique d'un graphe, les valeurs des variables associées peuvent être représentées dans un tableau.

4.6.5.8 Utilisation normalisée

Un GRAPH est appelé par un MENU ou une METHOD. Un GRAPH référence une ou plusieurs WAVEFORMs, chacune représentant une courbe unique sur le GRAPH. Chaque WAVEFORM décrit la source des points de données. L'application EDD exécute les INIT_ACTIONS avant d'afficher la WAVEFORM sur le GRAPH. Cela permet de procéder à des calculs des données

avant le rendu. Les REFRESH_ACTIONS sont exécutées par l'application EDD afin de spécifier la zone visible du GRAPH, ce qui permet une visualisation agrandie.

4.6.5.9 Intégration

Un GRAPH est rendu par une application EDD via un MENU ou une METHOD.

Un MENU peut contenir un ou plusieurs GRAPHS. Le graphique peut être intégré dans tous les menus visibles. Il est possible d'utiliser un ou plusieurs graphiques ensemble avec des champs d'entrée et de sortie dans un menu. L'application EDD exécute les INIT_ACTIONS avant de rendre le GRAPH.

Un GRAPH est rendu par une METHOD via des Builtins. Le Builtin MenuDisplay() est utilisé pour rendre le GRAPH. Les INIT_ACTIONS du GRAPH sont appelées pour afficher le menu.

4.6.5.10 GRAPH avec plusieurs WAVEFORMs qui font référence au même Y_AXIS

Une application EDD doit associer des WAVEFORMs faisant référence au même Y_AXIS. Les courbes doivent être regroupées dans la zone de graphique et un seul axe y doit être affiché sur un côté de la zone de graphique.

4.6.5.11 Formes d'onde et légendes multiples

Il convient que l'application EDD affiche une légende afin de distinguer plusieurs formes d'onde sur un même graphique. Le libellé de chaque forme d'onde est dérivé de l'attribut LABEL de la WAVEFORM. Il convient que les WAVEFORMs sans attribut LABEL ne s'affichent pas sur la légende.

4.6.5.12 Graphiques éditables

4.6.5.12.1 Généralités

L'attribut HANDLING d'une WAVEFORM détermine si l'utilisateur peut éditer la forme d'onde. Il convient que l'application EDD fournisse un mécanisme qui permet à l'utilisateur de modifier la forme d'onde (glisser-cliquer direct, entrée de tableau, etc.), ainsi qu'un mécanisme qui permet à l'utilisateur d'indiquer que la modification de la forme d'onde est achevée. Pour les GRAPH générés par un MENU, il convient que l'application EDD fournisse un mécanisme qui permet à l'utilisateur de commencer l'édition de la WAVEFORM. Il convient que l'application EDD mette à jour les données des WAVEFORM lorsque l'utilisateur indique que la modification de la forme d'onde est achevée (bouton d'enregistrement, par exemple). Il convient que l'application EDD fournisse un mécanisme qui permet à l'utilisateur de restaurer la WAVEFORM d'origine sans mettre à jour les données (bouton d'annulation, par exemple).

Pour les GRAPH qui possèdent une définition CYCLE_TIME, l'application EDD doit suspendre la valeur source lue à partir de l'appareil lorsque le GRAPH est en cours d'édition.

4.6.5.12.2 Attribut POST_EDIT_ACTIONS sur des variables

Il convient d'appeler les POST_EDIT_ACTIONS pour chaque modification d'une forme d'onde de graphique. Les entrées de données peuvent être contrôlées et les données peuvent être modifiées d'après une modification.

4.6.5.13 Légende et aide pour les courbes

L'EDD peut fournir des informations sur les libellés et les aides concernant les GRAPH et leurs courbes et axes. Afin de créer des légendes et des informations d'aide pour les courbes, l'application EDD doit prendre en charge les règles ci-après.

L'application EDD doit afficher la description si elle est définie comme une légende par les membres du GRAPH. L'application EDD doit utiliser le LABEL de la WAVEFORM référencée si la description n'est pas définie. Une chaîne vide constitue une chaîne définie.

L'application EDD doit donner les informations d'aide des membres du GRAPH aux utilisateurs.

Si aucun attribut HELP n'existe dans les membres du GRAPH, les informations d'aide WAVEFORM doivent être utilisées.

4.6.5.14 Fonctions d'agrandissement et de défilement prises en charge par l'appareil

Il convient que l'application EDD supporte les fonctions d'agrandissement et de défilement sans prise en charge dans l'EDD. L'application EDD affiche plus ou moins de points ou de données de la partie non affichée de la WAVEFORM sur la zone d'affichage. De plus, les REFRESH_ACTIONS dans l'EDD peuvent prendre en charge les fonctions d'agrandissement et de défilement. L'application EDD appelle les REFRESH_ACTIONS si l'utilisateur fait un défilement dans une zone pas stockée dans les données de forme d'onde. Elle appelle aussi les REFRESH_ACTIONS si l'utilisateur effectue un agrandissement et qu'il convient d'afficher plusieurs points. Les informations relatives à la position et à l'agrandissement peuvent être lues à partir de l'axe avec les attributs VIEW_MIN et VIEW_MAX.

La Figure 51 est un exemple d'EDD comportant des fonctions d'agrandissement et de défilement prises en charge par l'appareil.

```
VARIABLE      y_value
{
    LABEL "...";
    HELP "...";
    CLASS DYNAMIC;
    TYPE FLOAT;
    CONSTANT_UNIT [degC];
}

ARRAY          y_data
{
    NUMBER_OF_ELEMENTS 1000;
    TYPE y_value;
}

COMMAND read_data
{
    SLOT ...; INDEX...;
    OPERATION READ;
    TRANSACTION
    {
        REPLY
        {
            device_t_min_value, device_t_max_value,
            device_y_min_value, device_y_max_value,
            y_data
        }
    }
}

COMMAND read_current_data
{
    SLOT ...; INDEX...;
    OPERATION READ;
    TRANSACTION
    {
        REPLY
        {
            device_t_min_value, device_t_max_value,
            device_y_min_value, device_y_max_value,
            y_data
        }
    }
}
```

```

COMMAND write_data_area
{
    SLOT ...; INDEX...;
    OPERATION WRITE;
    TRANSACTION
    {
        REQUEST
        {
            t_min_value, t_max_value,
            y_min_value, y_max_value
        }
    }
}

VARIABLE y_increment
{
    LABEL "..."; HELP "...";
    CLASS LOCAL;
    TYPE TIME;
}

VARIABLE current_date_time1
{
    LABEL "..."; HELP "...";
    CLASS LOCAL;
    TYPE DATE_AND_TIME;
}

VARIABLE t_min_value
{
    LABEL "..."; HELP "...";
    CLASS LOCAL;
    TYPE DATE_AND_TIME;
}

t_max_value LIKE t_min_value;
t_device_min_value LIKE t_min_value;
t_device_max_value LIKE t_min_value;

VARIABLE y_min_value
{
    LABEL "..."; HELP "...";
    CLASS LOCAL;
    TYPE FLOAT;
}

y_max_value LIKE y_min_value;
device_y_min_value LIKE y_min_value;
device_y_max_value LIKE y_min_value;

AXIS y_axis_signature
{
    LABEL "...";
    HELP "...";
    MIN_VALUE y_min_value;
    MAX_VALUE y_max_value;
    SCALING LINEAR;
}

AXIS t_axis_signature
{
    LABEL "...";
    HELP "...";
    MIN_VALUE t_min_value;
    MAX_VALUE t_max_value;
    SCALING LINEAR;
}

WAVEFORM value_signature1
{
    LABEL "...";
    HELP "...";
    HANDLING READ; // alternative READ & WRITE
    LINE TYPE DATA1;
    EMPHASIS TRUE;

    TYPE XY
}

```

```

        X_INITIAL      current_date_time1; // starting time in milliseconds
        X_INCREMENT    y_increment;       // time between two points in ms
        Y_VALUES       { y_data }
    }

    Y_AXIS             y_axis_signature;

    INIT_ACTIONS { init_signature}
    REFRESH_ACTIONS { refresh_signature}
}

METHOD init_signature
{
    DEFINITION
    {
        // read current data and store area t_device_min_value,
        // t_device_max_value, y_device_min_value, y_device_max_value
        read_command (read_current_data);

        // calculation of the waveform data
        value_signature1.X_INITIAL = t_device_max_value;
        value_signature1.X_INCREMENT = (t_device_max_value-t_device_min_value)/
            ( y_data.NUMBER_OF_ELEMENTS - 1 );

        // set the visible area to the area from the device
        t_axis_signature.MIN_VALUE = t_device_min_value;
        t_axis_signature.MAX_VALUE = t_device_max_value;
        y_axis_signature.MIN_VALUE = y_device_min_value;
        y_axis_signature.MAX_VALUE = y_device_max_value;
    }
}

METHOD refresh_signature
{
    DEFINITION
    {
        // read data from the device depending of the current
        // MIN_VALUE and MAX_VALUE of the time axis
        t_min_value = t_axis_signature.MIN_VALUE;

        t_max_value = t_axis_signature.MAX_VALUE;
        y_min_value = y_axis_signature.MIN_VALUE;
        y_max_value = y_axis_signature.MAX_VALUE;

        // writing requested t_min_value, t_max_value, y_min_value, y_max_value
        write_command (write_data_area);

        // reading points and area t_device_min_value, t_device_max_value,
        // y_device_min_value, y_device_max_value from
the device
        read_command (read_data);

        // reduce the visible area to the from the device supported area
        if (t_device_min_value > t_min_value)
            t_axis_signature.MIN_VALUE = t_device_min_value;
        if (t_device_max_value < t_max_value)
            t_axis_signature.MAX_VALUE = t_device_max_value;
        if (y_device_min_value > y_min_value)
            y_axis_signature.MIN_VALUE = y_device_min_value;
        if (y_device_max_value < y_max_value)
            y_axis_signature.MAX_VALUE = y_device_max_value;
    }
}

```

Figure 51 – Exemple d'EDD avec fonctions d'agrandissement et de défilement prises en charge par l'appareil

4.6.6 AXIS

Les axes sont utilisés pour les diagrammes et les graphiques. Dans un diagramme comportant des courbes, l'axe x est contrôlé par l'application EDD. Dans un graphique, seul l'axe x peut être référencé.

Les axes y d'un diagramme sont référencés dans la SOURCE. Les axes y d'un graphique sont référencés dans la WAVEFORM. Si certaines sources ou formes d'onde font référence au

même axe y d'un diagramme ou graphique, il convient que l'application EDD ne trace l'axe y qu'une seule fois.

Les attributs MIN_VALUE et MAX_VALUE sont facultatifs. S'ils ne sont pas définis dans l'axe, l'application EDD les détermine de sorte que MIN_VALUE et MAX_VALUE doivent être en permanence calculés avec par exemple 150 % des données affichées (c'est-à-dire redimensionnées automatiquement). Dans le cas d'un graphique, l'application EDD peut générer les valeurs X et Y minimales et maximales des matrices. Dans le cas d'un diagramme, l'application EDD peut lire certaines valeurs et générer les valeurs minimales et maximales initiales. Il convient que l'application EDD recalcule les axes si la valeur n'est pas comprise dans la plage.

Les attributs VIEW_MIN et VIEW_MAX spécifient la zone d'affichage. L'application EDD procède à leur définition avant que les INIT_ACTIONS ne soient appelées et après que l'utilisateur ait modifié l'agrandissement ou le positionnement. Dans une méthode, les attributs VIEW_MIN et VIEW_MAX peuvent être lus et modifiés (par exemple, si l'utilisateur définit la position sur une zone comprise en dehors des données disponibles, la méthode peut les définir sur des valeurs existantes).

Le LABEL est utilisé pour spécifier la légende de l'axe. Si une CONSTANT_UNIT est définie, cette unité sera utilisée. Néanmoins, si la variable de l'axe est associée à une relation UNIT, cette unité est utilisée. Sinon, l'axe ne possède pas d'unité.

La SCALING d'un axe peut être LINEAR ou LOGARITHMIC. Par défaut, l'attribut SCALING est défini sur LINEAR.

L'application EDD affiche les horodatages absolus ou relatifs sur l'axe x d'un diagramme.

4.6.7 IMAGE

La construction de base IMAGE est utilisée pour l'affichage d'images dans des fenêtres, des boîtes de dialogue, des pages et des groupes. Si une application EDD ne peut pas afficher l'image faute d'espace d'affichage exigé, l'application EDD peut afficher le LABEL à la place de l'image. L'application EDD doit afficher les IMAGES qui ont de la transparence avec de la transparence par rapport à l'arrière-plan de son conteneur.

Pour les appareils portatifs, l'attribut PATH peut aussi posséder un code pays zz (voir 4.2).

La Figure 52 représente la définition d'une image.

```
IMAGE Sensor_Diagram
{
    LABEL "Sensor Diagram";
    HELP "This Diagram shows the sensor characteristic";
    PATH "SenDiaEn.jpg\
        |zz|SenDiaEn_LowRes.jpg\
        |de|SenDiaDe.jpg\
        |de zz|SenDiaDe_LowRes.jpg";
}
```

Figure 52 – Exemple d'EDD pour une IMAGE

Une définition IMAGE peut être utilisée pour appeler une METHOD ou un MENU STYLE DIALOG ou WINDOW avec l'attribut LINK.

La Figure 53 est un exemple d'EDD d'utilisation de l'attribut LINK dans une image.

```

MENU
{
    ITEMS
    {
        ...,
        Self_Test_Image (INLINE),
        ...
    }
}

IMAGE Self_Test_Image
{
    LABEL "Self Test";
    HELP "This Method execute the self test function of the device which needs some
time";
    PATH "SelfTestImage.jpg";
    LINK Self_Test_Menu;
}

MENU Self_Test_Menu
{
    ...
}
    
```

Figure 53 – Exemple d'EDD pour une IMAGE avec attribut LINK

Les formats d'image répertoriés dans le Tableau 7 peuvent être utilisés dans une image.

Tableau 7 – Formats d'image

Format	Description
JPG, JPEG	Format conforme aux exigences de l'ISO/IEC 10918
PNG	Format Portable Network Graphics conforme aux exigences de l'ISO/IEC 15948
GIF	Graphics Interchange Format

Plusieurs images localisées peuvent être spécifiées avec la même technique de localisation que celle utilisée avec les chaînes littérales.

EXEMPLE: CHEMIN "en|english.gif|de|german.gif"

Cet exemple spécifie un fichier d'image à utiliser lorsque l'application utilise l'anglais (english) et un autre lorsque l'allemand (german) est utilisé.

4.6.8 GRID

L'attribut GRID décrit une matrice de données provenant d'un appareil qui sont destinées à être affichées par l'application EDD. Une GRID est utilisée pour afficher des vecteurs de données avec l'en-tête ou la description des données dans ce vecteur. Les vecteurs sont affichés horizontalement (lignes) ou verticalement (colonnes) comme spécifié par l'attribut ORIENTATION.

La Figure 54 est un exemple de GRID.

```

VARIABLE PeakType
{
    LABEL "Peak Type";
    CLASS DEVICE;
    TYPE ENUMERATED
    {
        { 1, "False Echo"},
        { 2, "Button Echo" },
        { 3, "Unkown" }
    }
}
    
```

```

ARRAY arrPeakType
{
    NUMBER_OF_ELEMENTS 10;
    TYPE PeakType;
}

VARIABLE PeakDistance
{
    LABEL "Peak Distance";
    CLASS DEVICE;
    TYPE FLOAT
    {
        DEFAULT_VALUE 1.0;
    }
}

ARRAY arrPeakDistance
{
    LABEL "Peak Distance";
    NUMBER_OF_ELEMENTS 10;
    TYPE PeakDistance;
}

VARIABLE PeakAmplitude
{
    LABEL "Device Amplitude";
    CLASS DEVICE;
    TYPE FLOAT
    {
        DEFAULT_VALUE 1.0;
    }
}

ARRAY arrPeakAmplitude
{
    LABEL "Peak Amplitudes";
    NUMBER_OF_ELEMENTS 10;
    TYPE PeakAmplitude;
}

MENU DeviceEcho
{
    LABEL "Device Echo";
    ITEMS
    {
        EchoCurve,
        FoundEcho,
        FalseEchoes
    }
}

MENU FoundEcho
{
    LABEL "Found echoes";
    STYLE PAGE;
    ITEMS
    {
        GridFoundEcho,
        RegisterFalseEchoes
    }
}

GRID GridFoundEchoes
{
    LABEL "All detected echoes are displayed below";
    VALIDITY IF (varModelCode == 4711) { TRUE; } ELSE { FALSE; }
    VECTORS
    {
        {"Type", arrPeakType},
        {"Distance (m)", arrPeakDistance},
        {"Amplitude (mV)", arrPeakAmplitude}
    }
}

```

Figure 54 – Exemple d'EDD pour une GRID

La Figure 55 représente le résultat de l'exemple d'EDD ci-dessus.

Type	Distance (m)	Amplitude [mV]
False echo	1,331	626
False echo	2,385	831
Bottom echo	4,591	2874
Unknown	4,559	3821
Unknown	9,956	2218

Figure 55 – Résultat de l'exemple EDD

5 Description de données EDDL

5.1 Variables

5.1.1 TYPEs de VARIABLE

5.1.1.1 BIT_ENUMERATED

Les bits multiples peuvent être regroupés dans une variable énumérée à bit unique. Chaque bit pourrait avoir une signification distincte. Si une exécution forcée est exigée, il convient d'utiliser une variable ENUMERATED à la place d'une variable BIT_ENUMERATED.

La Figure 56 donne un exemple d'EDD pour illustrer la mauvaise utilisation d'une variable BIT_ENUMERATED.

```
VARIABLE Measuring_Mode
{
    LABEL " Measuring Mode";
    TYPE BIT_ENUMERATED
    {
        { 0x08, "Massflow"},
        { 0x11, "Flow" }      /* this is not referencing a single bit!!! */
    }                       /* This is not allowed */
}
```

Figure 56 – Mauvaise utilisation d'une variable BIT_ENUMERATED

Pour cet exemple, le type ENUMERATED est recommandé, car il n'est pas permis d'utiliser les deux en même temps.

La Figure 57 est un exemple d'EDD avec une VARIABLE de type ENUMERATED.

```
VARIABLE Measuring_Mode
{
    LABEL " Measuring Mode";
    TYPE ENUMERATED
    {
        { 8, "Massflow"},
        { 17, "Flow" }      /* this is correct */
    }
}
```

Figure 57 – Utilisation d'une variable ENUMERATED
à la place d'une variable BIT_ENUMERATED

5.1.1.2 ASCII, EUC, PACKED_ASCII, OCTET, PASSWORD, VISIBLE

L'application EDD doit afficher ces types de données sous forme de chaînes. En fonction du type des données, les caractères de queue doivent être traités comme représenté dans le Tableau 8.

Tableau 8 – Traitement de chaîne

TYPE	Terminaison	Caractère de remplissage	Suppression des espaces de queue
ASCII	Null	Null	Non
EUC	Null	Null	Non
PACKED_ASCII	Non terminé	Espace	Non
OCTET	Non terminé	Espace	Non
PASSWORD	HART: Null, FOUNDATION fieldbus: not terminated	HART: Null, FOUNDATION fieldbus: space	Non
VISIBLE	Non terminé	Espace	Oui

5.1.2 CLASSE de VARIABLE

5.1.2.1 Vue d'ensemble

L'attribut CLASS spécifie l'utilisation de la variable. Certaines CLASSES comportent des exigences spécifiques par rapport à l'application EDD.

5.1.2.2 CLASSE FACULTATIVE

Un profil d'appareil définit des variables comme étant obligatoires ou facultatives. Si un appareil prend en charge une fonctionnalité, il doit la prendre en charge de la manière définie par la variable facultative.

Si un appareil ne prend en charge aucune variable facultative, l'application EDD ne doit pas afficher la valeur de la variable. L'application EDD peut masquer l'ensemble de la variable dans ce cas.

Si une méthode lit une variable facultative non prise en charge, l'attribut VARIABLE_STATUS est défini sur VARIABLE_STATUS_NOT_SUPPORTED et la valeur de la variable n'existe pas. Avant que la méthode n'accède à la valeur de variable, la méthode doit contrôler l'attribut VARIABLE_STATUS sinon la méthode est interrompue.

Si une méthode écrit une variable facultative non prise en charge, l'attribut VARIABLE_STATUS est défini sur VARIABLE_STATUS_NOT_SUPPORTED.

5.1.3 ACTIONS de VARIABLE

5.1.3.1 PRE_EDIT_ACTIONS

Il convient d'utiliser les PRE_EDIT_ACTIONS pour afficher des informations ou des avertissements destinés à l'utilisateur. Il convient de ne pas les utiliser aux fins de manipulation de variables. Les méthodes sont appelées uniquement si l'utilisateur souhaite éditer la variable.

Si une METHOD des PRE_EDIT_ACTIONS est interrompue, il convient que l'application EDD signale à l'utilisateur l'existence d'un processus anormal et l'édition doit être annulée.

5.1.3.2 POST_EDIT_ACTIONS

Il convient d'utiliser les POST_EDIT_ACTIONS pour afficher des informations ou des avertissements ou pour manipuler des variables. Les méthodes sont appelées après que la valeur d'une variable a été modifiée.

Si une METHOD des POST_EDIT_ACTIONS est interrompue, il convient que l'application EDD signale à l'utilisateur l'existence d'un processus anormal.

5.1.3.3 REFRESH_ACTIONS

Les REFRESH_ACTIONS doivent être utilisées pour calculer la valeur de la variable à l'aide d'autres variables. Les REFRESH_ACTIONS ne doivent pas être utilisées pour modifier d'autres variables ou n'afficher aucune information. Les méthodes doivent être appelées systématiquement avant que la variable n'ait besoin d'être réactualisée dans des expressions conditionnelles, à des fins d'affichage, etc.

5.2 Données d'appareil stockées dans l'application EDDL

5.2.1 Vue d'ensemble

Plusieurs types d'appareils complexes ont besoin d'accéder aux données de performance historiques afin d'évaluer l'état de fonctionnement courant. Deux exemples illustrent ces types d'appareils: les positionneurs de vanne et les mesureurs de niveau radar.

Dans le cas d'un positionneur de vanne, une signature de référence est utilisée lorsque l'installation est achevée. Cette signature est comparée à la signature réelle à une date ultérieure. Des différences significatives entre les signatures peuvent indiquer qu'il est exigé de procéder à la maintenance.

Dans le cas d'un mesureur radar, plusieurs signaux retour peuvent être enregistrés sous différentes conditions d'exploitation (par exemple, différents niveaux de réservoir ou différents équipements d'exploitation). L'examen et/ou la comparaison de ces signaux peuvent permettre l'optimisation du fonctionnement du mesureur ou la détection d'un changement imprévu dans les conditions d'exploitation.

Le développeur EDD peut spécifier les données devant être stockées par l'application EDDL. Ces données peuvent être rappelées à une date ultérieure aux fins d'évaluation de la performance et/ou de l'état de l'appareil.

5.2.2 FILE

La construction FILE permet à un développeur EDD de spécifier les données devant être stockées pour une utilisation ultérieure. A l'instar d'une COLLECTION, elle ne fait pas de différence entre le référencement de données direct ou l'utilisation de la référence de fichier. Les deux constructions assurent l'accès aux mêmes données. La différence entre les constructions FILE et COLLECTION est que les données référencées par des FILES sont stockées.

Des données permanentes sont associées à une seule et même instance d'appareil. Envisager un système comportant quatre positionneurs de vanne identiques. La même EDD est utilisée pour les quatre appareils. Néanmoins, il existe quatre ensembles de données d'instance, un pour chaque positionneur de vanne. Lorsque la construction FILE est utilisée, l'application EDD stocke des données permanentes qui sont associées à chaque appareil. Si la construction FILE est utilisée pour stocker la signature de vanne de l'appareil, cette signature est alors disponible à une date ultérieure. La signature du positionneur de vanne n°2 n'est pas disponible lorsque l'EDD est utilisée par le positionneur de vanne n°1.

La construction FILE spécifie uniquement la structure des données à stocker. Elle ne spécifie pas la manière dont l'application EDD stocke les données, ni le moment où les données sont chargées dans la mémoire locale. L'application EDD peut charger les données du FILE au moment de l'instanciation de l'appareil ou peut fournir les données lorsque l'EDD en fait la

demande. Les données permanentes peuvent être stockées dans une mémoire flash ou sur un disque dur, sur l'ordinateur exécutant l'application EDD ou sur un serveur de fichiers. Le FILE peut être stocké sous la forme d'un fichier plat dans la structure de fichiers des systèmes d'exploitation ou peut être embarqué à la base de données de l'application EDD. Dans tous les cas, l'accès à l'ensemble des données du FILE est disponible une fois l'EDD instancié. L'EDD n'a pas besoin d'appeler des commandes d'ouverture, de fermeture, de lecture ou d'écriture des fichiers de niveau inférieur.

Le développeur EDD définit le schéma et la structure de données du FILE. Le FILE comporte une liste de membres pouvant être des combinaisons d'éléments de données, par exemple VARIABLES, ARRAYS, RECORDS, COLLECTIONS ou LISTS. Cette flexibilité permet au développeur EDD de stocker un ensemble fixe de données uniquement avec des VARIABLES, des ARRAYS, des RECORDS et des COLLECTIONS. Le développeur EDD peut également stocker une quantité variable de données à l'aide de la construction LIST (conjointement aux constructions VARIABLE, ARRAY, RECORD et COLLECTION).

La Figure 58 est un exemple simple de déclaration FILE. Dans ce cas, une demande de prise en charge d'un fichier appelé "reference_valve_signature" est soumise à l'application EDDL.

```
VARIABLE valve_position
{
  TYPE FLOAT;
}

ARRAY valve_stroke
{
  TYPE valve_position;
  NUMBER_OF_ELEMENTS 1000;
}

VARIABLE valve_signature_comment
{
  TYPE ASCII(64);
}

FILE reference_valve_signature
{
  MEMBERS
  {
    COMMENT, valve_signature_comment;
    STROKE, valve_stroke;
  }
}
```

Figure 58 – Exemple de déclaration de fichier

Ce FILE comporte une petite note ("valve_signature_comment") sur la signature, et la signature proprement dite ("valve_stroke"). La signature est une matrice de 1 000 échantillons de position de vanne également espacés.

Il s'agit d'un exemple simple. Dans une application réelle, les échantillons de position peuvent ne pas être également espacés, ce qui nécessitera un temps d'échantillonnage. Si tel est le cas, une autre matrice de temps d'échantillonnage sera nécessaire. D'autres informations complémentaires (date, heure, opérateur, etc.) peuvent être ajoutées.

L'accès aux membres d'un FILE se fait à l'aide de la notation à point comme pour le référencement d'un membre COLLECTION. Ainsi

```
reference_valve_signature.STROKE[10];
```

accéderait à la 10^e valeur dans la matrice de signature, ou valve_stroke[10]. Il est également simple de tracer cette signature et de la comparer à la signature réelle. Bien que cela soit normalement réalisé par une méthode, on peut également le faire via un MENU.

La Figure 59 représente un GRAPH dans un MENU.

```

ARRAY current_stroke { TYPE valve_position; NUMBER_OF_ELEMENTS 1000; }

VARIABLE sample_interval { TYPE FLOAT; CONSTANT_UNITS "ms"; }

WAVEFORM stroke_now
{
    TYPE YT
    {
        X_INITIAL 0;
        X_INCREMENT sample_interval;
        Y_VALUES {current_stroke }
    }
}

WAVEFORM ref_stroke
{
    TYPE YT
    {
        X_INITIAL 0;
        X_INCREMENT sample_interval;
        Y_VALUES { valve_stroke }
    }
}

GRAPH compare_stroke
{
    MEMBERS
    {
        NOW, stroke_now;
        THEN, ref_stroke;
    }
}

MENU compare_strokes
{
    LABEL "CompStrokes";
    ITEMS
    {
        compare_stroke
    }
}

```

Figure 59 – Exemple de comparaison des signatures d'une vanne

Dans l'exemple de la Figure 59, les données échantillonnées lors d'une course récente de la vanne sont stockées dans l'attribut "current_stroke" et l'intervalle d'échantillonnage (l'inverse du taux d'échantillonnage) indique l'espacement temporel entre les points d'échantillonnage. Deux WAVEFORM et un GRAPH sont déclarés, ainsi qu'un MENU qui contient le GRAPH. Le GRAPH sera affiché par l'application EDDL lorsque le MENU est accédé.

Lorsque le graphique est affiché, l'application EDD reconnaît que l'attribut "valve_stroke" est un membre d'un FILE. Par conséquent, lorsque le GRAPH est affiché, il convient que les données soient automatiquement fournies à partir du fichier "reference_valve_signature".

5.2.3 LIST

La construction LIST est une matrice insérable de longueur variable. Chaque élément stocké dans la liste a exactement la même structure. Cette structure est spécifiée au moyen de l'attribut TYPE obligatoire. L'attribut TYPE peut faire référence à une structure ou à un élément de données EDD légal (par exemple, VARIABLE, ARRAY, RECORD, COLLECTION, LIST). L'accès aux éléments individuels de la LIST se fait à l'aide de la notation entre crochets, comme pour l'accès des éléments ARRAY.

Un exemple d'utilisation d'une construction LIST dans une construction FILE est donné dans la Figure 60. Dans cet exemple, la taille de la construction LIST (et donc la construction FILE) peut augmenter au fur et à mesure que sont stockées des formes d'onde radar significatives complémentaires qui documentent le fonctionnement du mesureur de niveau.

```

VARIABLE gauge_location { TYPE ASCII(32); }
VARIABLE tag { TYPE ASCII(32); }

VARIABLE date_taken { TYPE DATE; }
VARIABLE operator { TYPE ASCII(32); }
VARIABLE operating_conditions { TYPE ASCII(64); }
VARIABLE sample_interval { TYPE FLOAT; CONSTANT_UNITS "ms"; }

VARIABLE sample { TYPE UNSIGNED_INTEGER(2); }
ARRAY echo_signal { TYPE sample; NUMBER_OF_ELEMENTS 4096; }

COLLECTION signal_info
{
    MEMBERS
    {
        DATE_STAMP, date_taken;
        TAKEN_BY, operator;
        NOTES, operating_conditions;
        DT, sample_interval;
        SIGNAL, echo_signal;
    }
}

LIST recorded_signals { TYPE signal_info; }

FILE stored_signals
{
    MEMBERS
    {
        LOCATION, gauge_location;
        INSTR_TAG, tag;
        SIGNALS, recorded_signals;
    }
}

```

Figure 60 – Exemple d'une déclaration de fichier plus complexe

Dans l'exemple de la Figure 60, des informations de base ("gauge_location", "tag") concernant le mesureur de niveau sont stockées avec une série de signaux radar ("recorded_signals"). Dans cet exemple, "signal_info" est utilisé comme définition de type pour la liste. Le référencement de "signal_info" dans l'EDD n'accédera pas à la liste. L'accès à la liste peut uniquement être fait à l'aide de la notation entre crochets. Les deux références suivantes se rapportent aux mêmes données.

```

stored_signals.SIGNALS[10]
recorded_signals[10]

```

Toutefois, les références suivantes n'accèdent pas aux mêmes informations.

```

stored_signals.SIGNALS[10].SIGNAL
echo_signal;

```

La référence "echo_signal" définit la structure correspondant à une partie d'un élément de liste et, dans les deux cas, la quantité de données référencées est la même (dans les deux cas une matrice de 4 096 entiers non signés à 2 octets). Toutefois, la valeur "echo_signal" sera nulle tant que des données ne seront pas placées dedans (en chargeant des données à partir de l'appareil de terrain, par exemple).

La Figure 61 est un exemple de méthode utilisée pour l'examen des signaux radar stockés. Dans cet exemple, la construction LIST des formes d'onde stockées s'affiche de manière séquentielle. La variable "i" est utilisée pour exécuter pas à pas l'ensemble de la LIST, à l'instar d'un itérateur.

```

WAVEFORM one_echo
{
    TYPE YT
    {
        X_INITIAL 0;
        X_INCREMENT sample_interval;
        Y_VALUES { echo_signal }
    }
}

GRAPH review_radar_signal
{
    MEMBERS
    {
        PING, one_echo;
    }
}

MENU review_radar_signal_menu
{
    LABEL "Radar Signal";
    STYLE WINDOW;
    ITEMS
    {
        review_radar_signal
    }
}

/*
*****
* now for a method that reviews the stored radar signals (ping).
*/
METHOD reviewStoredPings
{
    LABEL "SignalHistory";
    DEFINITION
    {
        int i,
        ans; /*The users answer to select from list*/
        long selection;
        long varid[5]; /*used in the acknowledge Builtin calls*/

        i = 0;

        varid[0] = VARID( date_taken );
        varid[1] = VARID( operator );
        varid[2] = VARID( operating_conditions );
        varid[3] = VARID( sample_interval );
        varid[4] = VARID( echo_signal );

        do
        {
            /* get the current record so we can display it */
            date_taken = stored_signals.SIGNALS[i].DATE_STAMP;
            operator = stored_signals.SIGNALS[i].TAKE_BY;
            operating_conditions = stored_signals.SIGNALS[i].NOTES;
            sample_interval = stored_signals.SIGNALS[i].DT;
            echo_signal = stored_signals.SIGNALS[i].SIGNAL;

            /* display the graph and the annotations. */

            MenuDisplay (review_radar_signal_menu,"OK", selection);
            acknowledge ( "Radar Signal by %1} \nDescription %2}" varid);

            ans = select_from_list ("do you want to see the next radar
signal",
                                "Yes;No");

            if (ans != 0)
            {
                break;
            }
            i++;
        } while ( i < recorded_signals.COUNT);
    }
}

```

Figure 61 – Exemple d'examen des signaux radar stockés

Cet exemple a de nombreuses fonctionnalités significatives. D'abord, le Builtin MenuDisplay permet d'utiliser des menus STYLE WINDOW ou DIALOG à utiliser au sein des méthodes.

Chaque itération appelle également la méthode "acknowledge()". Dans une méthode, l'objet graphique est séparé et affiché de manière asynchrone. Cela permet au développeur EDD d'utiliser les Builtins "acknowledge" (delay et put_message) pour interagir avec l'utilisateur exactement de la même manière que celle toujours utilisée dans les METHODS.

L'appel de la méthode "acknowledge" affiche l'opérateur et les commentaires concernant le signal enregistré. La date n'est pas affichée. On peut afficher la date après quelques ajustements, mais la mise en œuvre n'est pas représentée dans cet exemple.

Enfin, il convient de noter que le COUNT "recorded_signals.COUNT" (ainsi que les attributs FIRST, LAST et CAPACITY) est un attribut inhérent à toutes les constructions LIST. L'attribut COUNT peut être utilisé pour détecter la fin de l'élément LIST. L'accès à des éléments LIST non existants dans des METHODS entraîne l'interruption du processus.

Si l'attribut COUNT est spécifié dans une LIST, il définit la taille des données d'appareil d'instance lors de sa création. S'il n'est pas défini, la liste est vide.

Pour remplir une liste dans une méthode, on peut appeler le Builtin ListInsert ou procéder à la lecture à l'aide d'une commande avec une variable avec les attributs INDEX ou INFO. Pour supprimer les éléments d'une liste dans une méthode, on peut appeler ListDeleteElementAt.

L'index d'une liste débute à 0 et est toujours continu. Si un élément d'une liste est supprimé, les index des éléments suivants seront décrémentés. Si un élément est inséré, les index des éléments suivants seront incrémentés.

L'attribut COUNT peut être utilisé pour obtenir le nombre d'éléments dans une liste. Il doit automatiquement être mis à jour avec ListInsert ou ListDeleteElementAt par l'application EDD.

La Figure 62 est plus impliquée. L'attribut procède à une mesure et relit le signal radar aux fins d'examen par l'opérateur. La commande 128 lance une mesure tandis que la commande 129 lit le signal renvoyé par l'appareil de terrain. Une fois l'opération terminée, le signal est affiché à l'utilisateur. L'utilisateur peut procéder à une autre lecture si celle-ci n'est pas satisfaisante.

Une fois une lecture significative trouvée, on peut l'ajouter au FILE (fichier) "stored_signals", remplacer un signal existant stocké dans des "stored_signals" ou comparer la lecture à celle d'un signal stocké. Dans de nombreux cas, les signaux stockés sont exécutés pas à pas comme dans l'exemple précédent.

Dans le cas d'une insertion, l'utilisateur peut insérer les nouvelles données au premier plan, au dernier plan ou au centre de la LIST (liste) contenue dans le FILE (fichier) "stored_signals". Cette partie de code illustre l'utilisation de l'attribut COUNT et de la fonction Builtin ListInsert().

La partie exécutant la fonction de remplacement ordonne la liste afin de trouver l'élément à remplacer. Un appel du Builtin ListDeleteElementAt() suivi d'un appel ListInsert().call affecte les remplacements.

Dans la dernière partie de l'exemple, le signal radar est comparé à un signal stocké. La liste est exécutée pas à pas et le GRAPH "compare_radar_signals" affiche les deux signaux sur le même graphique.

```

VARIABLE ping_index      { TYPE INDEX ping_data; }
ARRAY ping_data          { TYPE sample; NUMBER_OF_ELEMENTS 4096; }
VARIABLE today           { TYPE DATE; }
VARIABLE user            { TYPE ASCII(32); }
VARIABLE conditions      { TYPE ASCII(64); }

COLLECTION liveSig
{
    MEMBERS
    {
        DATE_STAMP, today;
        TAKEN_BY, user;
        NOTES, conditions;
        DT, sample_interval;
        SIGNAL, pind_data;
    }
}

COMMAND ping
{
    NUMBER 128;
    OPERATION COMMAND;

    TRANSACTION
    {
        REQUEST { }
        REPLY { response_code, device_status }
    }
    RESPONSE_CODES { ... }
}

COMMAND read_ping_data
{
    NUMBER 129;
    OPERATION READ;

    TRANSACTION
    {
        REQUEST { ping_index (INFO, INDEX) }
        REPLY
        {
            response_code, device_status, ping_index,
            ping_data[ping_index+00], ping_data[ping_index+01],
            ping_data[ping_index+02], ping_data[ping_index+03],
            ping_data[ping_index+04], ping_data[ping_index+05],
            ping_data[ping_index+06], ping_data[ping_index+07],
            ping_data[ping_index+08], ping_data[ping_index+09],
            ping_data[ping_index+10], ping_data[ping_index+11],
            ping_data[ping_index+12], ping_data[ping_index+13],
            ping_data[ping_index+14], ping_data[ping_index+15]
        }
    }
    RESPONSE_CODES { ... }
}

WAVEFORM new_echo
{
    TYPE YT
    {
        X_INITIAL 0;
        X_INCREMENT sample_interval;
        Y_VALUES { echo_signal }
    }
}

GRAPH new_radar_signal
{
    MEMBERS
    {
        PING, new_echo;
    }
}

GRAPH compare_radar_signals
{
    MEMBERS
    {
        PING1, new_echo;
    }
}

```

```

        PING2, one_echo;
    }
}

/*
*****
* pings the radar and captures the echo waveform into ping_data
*/
#define PING_COMPLETE 0x10;

METHOD newPing
{
    LABEL "Ping";
    DEFINITION
    {
        int i,
            ans;          /*The users answer to select from list*/
        long selector;
        long varid[5];    /*used in the acknowledge Builtin calls*/

        i = 0;

        varid[0] = VARID( date_taken );
        varid[1] = VARID( operator );
        varid[2] = VARID( operating_conditions );
        varid[3] = VARID( sample_interval );
        varid[4] = VARID( echo_signal );

        /*
        * ping once to get a radar signal
        */
        do {
            select_from_list ("Ready to make a measurement","Yes;No",ans);
            while (ans == 0) {
                send(128);          /* ping */
                do {
                    /* check the status of the ping */
                    send(48);
                } while(GET_VAR_VALUE (xmtr_specific_status4) & PING_COMPLETE);
            }
            /*
            * ping complete, read the signal
            */
            for (ping_index = 0; ping_index < 4096; ping_index += 16) {
                send (129);
            }
            MenuDisplay (new_radar_signal_menu, "OK", selector);
            /*
            * assumes the graph is displayed until I destroy it
            */
            select_from_list ("Take more radar data","Yes;No",ans);
        }
        /*
        * radar signal looks good. save it?
        */
        select_from_list("what now?", "save the signal;" \
                        "replace a stored signal; compare signals;" \
                        "get new signal; quit", ans);

        switch (ans)
        {
            case 0:          /* save the signal */
                get_dev_var_value (conditions);
                select_from_list("save where?","front of list; end of list;
                                insert into list", ans);

                switch (ans) {
                    case 0: i = 0; break;          /* beginning */
                    case 1: i = recorded_signals.COUNT; break; /* end */

                    default:          /* insert */
                        i = 0;
                        do {
                            /* get the current record so we can display it */
                            operator = stored_signals.SIGNALS[i].TAKE_BY;
                            operating_conditions =
                                stored_signals.SIGNALS[i].NOTES;
                            sample_interval = stored_signals.SIGNALS[i].DT;
                            echo_signal = stored_signals.SIGNALS[i].SIGNAL;

```

```

        /* display the graph and the annotations. */
        MenuDisplay (review_radar_signal_menu,
"OK", selector);
        acknowledge ( "Radar Signal by %{1} \n" \
            "Description %{2}" varid);

        select_from_list ("Insert here?","Yes;No",ans);

        if (ans == 0) {
            break;
        }
        i++;
    } while ( i < recorded_signals.COUNT);
    break;

}
ListInsert ( storedSignals.SIGNALST, i, liveSig );
break;

case 1:          /* replace a stored signal */
i = 0;
do {
    /* get the current record so we can display it */
    operator          = stored_signals.SIGNALS[i].TAKE_BY;
    operating_conditions= stored_signals.SIGNALS[i].NOTES;
    sample_interval    = stored_signals.SIGNALS[i].DT;
    echo_signal        = stored_signals.SIGNALS[i].SIGNAL;

    /* display the graph and the annotations. */

    MenuDisplay (review_radar_signal_menu, "OK", selector);
    acknowledge ( "Radar Signal by %{1} \n" \
        "Description %{2}" varid);
    select_from_list ("replace signal?","Yes;No",ans);

    if (ans == 0) {
        ListDeleteElement ( storedSignals.SIGNALST, i );
        ListInsert ( storedSignals.SIGNALST, i, liveSig );
        break;
    }
    i++;
} while ( i < recorded_signals.COUNT);
acknowledge("no signals replaced");
break;

case 2:          /* compare signals */
i = 0;
do {
    /* get the current record so we can display it */
    operator          = stored_signals.SIGNALS[i].TAKE_BY;
    operating_conditions= stored_signals.SIGNALS[i].NOTES;
    sample_interval    = stored_signals.SIGNALS[i].DT;
    echo_signal        = stored_signals.SIGNALS[i].SIGNAL;

    /* display the graph and the annotations. */

    MenuDisplay (review_radar_signal_menu, "OK", selector);
    acknowledge ( "Radar Signal by %{1} \n" \
        "Description %{2}" varid);
    select_from_list ("compare with this signal?",
        "Yes;No",ans);

    if (ans == 0) {
        MenuDisplay ( compare_radar_signal_menu, "OK", selector);
        select_from_list ("compare with another
            signal?","Yes;No",ans);

        if (ans==0) {
            continue;
        }
        break;
    }
    i++;
} while ( i < recorded_signals.COUNT);
acknowledge("no signals compared");
break;

```

```

        case 3:          /* get new signal */
            continue;

        default:         /* quit */
            process_abort ();
            break;
    }
} while ( 1 );
}

```

Figure 62 – Exemple d'EDD qui insère, remplace ou compare des signaux radar

5.3 Exposition des éléments de données en dehors de l'application EDD

Tous les éléments de données dont VALIDITY est égal à FALSE ne doivent pas être exposés.

L'application EDD ne doit pas exposer les éléments de données en dehors de l'application EDD si l'attribut PRIVATE est évalué comme TRUE. Dans le cas d'un élément COLLECTION, RECORD ou REFERENCE_ARRAY où l'attribut PRIVATE est évalué comme FALSE ou n'est pas défini, l'application EDD doit exposer l'élément COLLECTION, RECORD ou REFERENCE_ARRAY, ainsi que l'ensemble des éléments référencés, quel que soit l'attribut PRIVATE des éléments de données référencés.

Dans le cas d'un élément COLLECTION, RECORD ou REFERENCE_ARRAY où l'attribut PRIVATE est évalué comme TRUE, l'élément COLLECTION, RECORD ou REFERENCE_ARRAY n'est pas exposé, mais les éléments de données référencés peuvent être exposés au moyen d'autres règles.

5.4 Initialisation d'instances EDD

5.4.1 Vue d'ensemble

L'initialisation de variables est importante pour la configuration hors ligne. Si un appareil est configuré, lors de la phase de planification par exemple, la configuration peut être réalisée en cohérence avec les appareils. Le développeur EDD définit des plages et des énumérations avec des expressions conditionnelles pour permettre à l'application EDD de vérifier la cohérence des données et d'éviter la survenue de problèmes pendant le téléchargement descendant de la configuration.

5.4.2 Prise en charge de l'initialisation

Si l'utilisateur choisit un appareil, une instance est créée par l'application EDD avec l'EDD associée. Les variables EDD doivent être initialisées à l'aide des règles suivantes:

- Variables initialisées avec la valeur initiale de type variable. Cette règle doit s'appliquer si les points b), c) et d) ne s'appliquent pas.
- Initialisation des variables avec EDDL INITIAL_VALUES. Cette règle doit s'appliquer si le point a) ne s'applique pas.
- Initialisation des variables avec DEFAULT_VALUES. Cette règle doit s'appliquer si les INITIAL_VALUES n'existent pas.
- Initialisation de variables avec EDDL COMPONENT.

L'utilisateur peut en outre choisir un TEMPLATE avec d'autres initialisations. FOUNDATION fieldbus, PROFIBUS et PROFINET peuvent posséder des fichiers complémentaires pour l'EDD qui peuvent définir des valeurs par défaut.

5.4.3 TEMPLATE

Les TEMPLATES spécifient des valeurs de paramètre par défaut pour différentes utilisations d'un appareil; autrement dit, ils sont spécifiques à l'application. Les TEMPLATES peuvent être spécifiés par le fabricant de l'appareil dans l'EDD. Par exemple, un appareil de pression