

INTERNATIONAL STANDARD

NORME INTERNATIONALE



**Function blocks (FB) for process control –
Part 3: Electronic Device Description Language (EDDL)**

**Blocs fonctionnels (FB) pour les procédés industriels –
Partie 3: Langage de description électronique de produit (EDDL)**



THIS PUBLICATION IS COPYRIGHT PROTECTED

Copyright © 2010 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 la CEI ou du Comité national de la CEI du pays du demandeur.

Si vous avez des questions sur le copyright de la CEI ou si vous désirez obtenir des droits supplémentaires sur cette publication, utilisez les coordonnées ci-après ou contactez le Comité national de la CEI de votre pays de résidence.

IEC Central Office
3, rue de Varembe
CH-1211 Geneva 20
Switzerland
Email: inmail@iec.ch
Web: 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.

- Catalogue of IEC publications: www.iec.ch/searchpub

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 It also gives information on projects, withdrawn and replaced publications.

- IEC Just Published: www.iec.ch/online_news/justpub

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

- Electropedia: www.electropedia.org

The world's leading online dictionary of electronic and electrical terms containing more than 20 000 terms and definitions in English and French, with equivalent terms in additional languages. Also known as the International Electrotechnical Vocabulary online.

- Customer Service Centre: www.iec.ch/webstore/custserv

If you wish to give us your feedback on this publication or need further assistance, please visit the Customer Service Centre FAQ or contact us:

Email: csc@iec.ch

Tel.: +41 22 919 02 11

Fax: +41 22 919 03 00

A propos de la CEI

La Commission Electrotechnique Internationale (CEI) 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 CEI

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

- Catalogue des publications de la CEI: www.iec.ch/searchpub/cur_fut-f.htm

Le Catalogue en-ligne de la CEI vous permet d'effectuer des recherches en utilisant différents critères (numéro de référence, texte, comité d'études,...). Il donne aussi des informations sur les projets et les publications retirées ou remplacées.

- Just Published CEI: www.iec.ch/online_news/justpub

Restez informé sur les nouvelles publications de la CEI. Just Published détaille deux fois par mois les nouvelles publications parues. Disponible en-ligne et aussi par email.

- Electropedia: www.electropedia.org

Le premier dictionnaire en ligne au monde de termes électroniques et électriques. Il contient plus de 20 000 termes et définitions en anglais et en français, ainsi que les termes équivalents dans les langues additionnelles. Egalement appelé Vocabulaire Electrotechnique International en ligne.

- Service Clients: www.iec.ch/webstore/custserv/custserv_entry-f.htm

Si vous désirez nous donner des commentaires sur cette publication ou si vous avez des questions, visitez le FAQ du Service clients ou contactez-nous:

Email: csc@iec.ch

Tél.: +41 22 919 02 11

Fax: +41 22 919 03 00

INTERNATIONAL STANDARD

NORME INTERNATIONALE



**Function blocks (FB) for process control –
Part 3: Electronic Device Description Language (EDDL)**

**Blocs fonctionnels (FB) pour les procédés industriels –
Partie 3: Langage de description électronique de produit (EDDL)**

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

COMMISSION
ELECTROTECHNIQUE
INTERNATIONALE

PRICE CODE
CODE PRIX

XH

ICS 25.040.40; 35.240.50

ISBN 978-2-88912-560-9

CONTENTS

FOREWORD.....	19
INTRODUCTION.....	22
1 Scope.....	23
2 Normative references	23
3 Terms, definitions, abbreviated terms and acronyms	24
3.1 Terms and definitions	24
3.2 Abbreviated terms and acronyms	29
4 Conformance statement.....	30
5 Conventions for lexical structures	30
6 EDD and EDDL model	32
6.1 Overview of EDD and EDDL	32
6.2 EDD architecture	32
6.3 Concepts of EDD.....	32
6.4 Principles of the EDD development process	32
6.5 Interrelations between the lexical structure and formal definitions	33
6.6 Builtins	33
6.7 Profiles.....	34
7 Electronic Device Description Language	34
7.1 Overview	34
7.2 EDD identification information	42
7.3 AXIS.....	45
7.4 BLOCK.....	47
7.5 CHART.....	57
7.6 COLLECTION.....	59
7.7 COMMAND	60
7.8 COMPONENT	66
7.9 COMPONENT_FOLDER.....	70
7.10 COMPONENT_REFERENCE.....	70
7.11 COMPONENT_RELATION	71
7.12 CONNECTION	74
7.13 DOMAIN.....	75
7.14 EDIT_DISPLAY	75
7.15 FILE	77
7.16 GRAPH	77
7.17 GRID.....	78
7.18 IMAGE	80
7.19 IMPORT	81
7.20 INTERFACE	95
7.21 LIKE.....	96
7.22 LIST	96
7.23 MENU	98
7.24 METHOD.....	104
7.25 PROGRAM	106
7.26 RECORD.....	107
7.27 REFERENCE_ARRAY.....	107
7.28 Relations.....	108

7.29 RESPONSE_CODES.....	109
7.30 SOURCE.....	110
7.31 TEMPLATE.....	111
7.32 VALUE_ARRAY.....	112
7.33 VARIABLE.....	113
7.34 VARIABLE_LIST.....	129
7.35 WAVEFORM.....	129
7.36 Common attributes.....	135
7.37 Conditional expression.....	142
7.38 Referencing.....	143
7.39 Strings.....	152
7.40 Expression.....	154
7.41 Text dictionary.....	158
Annex A (normative) EDDL formal definition.....	160
Annex B (normative) EDDL Builtin library.....	246
Annex C (informative) EDD example.....	396
Annex D (normative) Profiles of EDDL and Builtins.....	410
Annex E (informative) Historical background.....	442
Bibliography.....	443
Figure 1 – Position of the IEC 61804 series related to other standards and products.....	22
Figure 2 – EDD generation process.....	33
Figure 3 – BLOCK_A.....	35
Figure 4 – CHART.....	35
Figure 5 – COLLECTION.....	36
Figure 6 – COMMAND.....	36
Figure 7 – DOMAIN.....	36
Figure 8 – EDIT_DISPLAY.....	37
Figure 9 – FILE.....	37
Figure 10 – GRAPH.....	37
Figure 11 – GRID.....	37
Figure 12 – IMAGE.....	37
Figure 13 – LIKE.....	38
Figure 14 – LIST.....	38
Figure 15 – MENU.....	38
Figure 16 – PROGRAM.....	39
Figure 17 – RECORD.....	39
Figure 18 – REFERENCE_ARRAY.....	39
Figure 19 – REFRESH.....	40
Figure 20 – UNIT.....	40
Figure 21 – WRITE_AS_ONE.....	40
Figure 22 – SOURCE.....	40
Figure 23 – VALUE_ARRAY.....	40
Figure 24 – VARIABLE.....	41
Figure 25 – VARIABLE_LIST.....	41

Figure 26 – WAVEFORM	41
Figure 27 – EDDL import mechanisms	81
Figure 28 – MENU activation	104
Figure 29 – Time for read-and-write operation	128
Figure C.1 – Example of an operator screen using EDD.....	396
Table 1 – Field attribute descriptions	31
Table 2 – DD_REVISION attribute	43
Table 3 – DEVICE_REVISION attribute.....	43
Table 4 – DEVICE_TYPE attribute	44
Table 5 – EDD_PROFILE attribute.....	44
Table 6 – EDD_VERSION attribute	44
Table 7 – MANUFACTURER attribute	45
Table 8 – MANUFACTURER_EXT attribute.....	45
Table 9 – AXIS attributes	45
Table 10 – MAX_VALUE, MIN_VALUE attribute.....	46
Table 11 – SCALING attribute.....	47
Table 12 – BLOCK_A attributes	47
Table 13 – CHARACTERISTIC attribute.....	48
Table 14 – PARAMETER attributes.....	48
Table 15 – AXIS_ITEMS attribute	49
Table 16 – CHART_ITEMS attribute.....	49
Table 17 – COLLECTION_ITEMS attribute	49
Table 18 – EDIT_DISPLAY_ITEMS attribute	50
Table 19 – FILE_ITEMS attribute.....	50
Table 20 – GRAPH_ITEMS attribute	50
Table 21 – GRID_ITEMS attribute.....	50
Table 22 – IMAGE_ITEMS attribute	51
Table 23 – LIST_ITEMS attribute	51
Table 24 – MENU_ITEMS attribute	51
Table 25 – METHOD_ITEMS attribute.....	52
Table 26 – PARAMETER_LISTS attributes	52
Table 27 – REFERENCE_ARRAY_ITEMS attribute.....	52
Table 28 – REFRESH_ITEMS attribute	52
Table 29 – SOURCE_ITEMS attribute.....	53
Table 30 – UNIT_ITEMS attribute	53
Table 31 – WAVEFORM_ITEMS attribute	53
Table 32 – WRITE_AS_ONE_ITEMS attribute.....	54
Table 33 – CHARTS attribute.....	54
Table 34 – LISTS attribute	54
Table 35 – GRAPHS attribute	55
Table 36 – GRIDS attribute.....	55
Table 37 – MENUS attribute	55

Table 38 – METHODS attribute	56
Table 39 – BLOCK_B attributes	56
Table 40 – NUMBER attributes	56
Table 41 – TYPE attributes	57
Table 42 – CHART attributes	57
Table 43 – CYCLE_TIME attribute	58
Table 44 – LENGTH attribute	58
Table 45 – TYPE attribute	58
Table 46 – COLLECTION attributes	59
Table 47 – item-type	59
Table 48 – COMMAND attributes	61
Table 49 – OPERATION attribute	61
Table 50 – TRANSACTION attributes	62
Table 51 – REPLY and REQUEST attributes	62
Table 52 – INDEX attribute	64
Table 53 – BLOCK_B attribute	64
Table 54 – NUMBER attribute	64
Table 55 – SLOT attribute	65
Table 56 – SUB_SLOT attribute	65
Table 57 – CONNECTION attribute	65
Table 58 – HEADER attribute	66
Table 59 – MODULE attribute	66
Table 60 – COMPONENT attribute	66
Table 61 – CAN_DELETE attribute	67
Table 62 – CHECK_CONFIGURATION attribute	68
Table 63 – COMPONENT_RELATIONS attribute	68
Table 64 – DECLARATION attribute	68
Table 65 – DETECT attribute	68
Table 66 – EDD attribute	69
Table 67 – CHECK_CONFIGURATION attribute	69
Table 68 – REDUNDANCY attribute	69
Table 69 – SCAN attribute	70
Table 70 – SCAN_LIST attribute	70
Table 71 – COMPONENT_FOLDER attribute	70
Table 72 – COMPONENT_REFERENCE attribute	71
Table 73 – COMPONENT_RELATION attribute	71
Table 74 – COMPONENTS attribute	72
Table 75 – RELATION_TYPE attribute	72
Table 76 – ADDRESSING attribute	73
Table 77 – MAXIMUM_NUMBER attribute	73
Table 78 – MINIMUM_NUMBER attribute	73
Table 79 – REQUIRED_INTERFACE attribute	74
Table 80 – CONNECTION attribute	74

Table 81 – APPINSTANCE attribute.....	74
Table 82 – DOMAIN attributes	75
Table 83 – EDIT_DISPLAY attributes.....	75
Table 84 – EDIT_ITEMS attribute	76
Table 85 – DISPLAY_ITEM attributes	76
Table 86 – POST_EDIT_ACTIONS, PRE_EDIT_ACTIONS attribute.....	77
Table 87 – FILE attributes.....	77
Table 88 – GRAPH attributes.....	78
Table 89 – CYCLE_TIME attribute	78
Table 90 – X_AXIS attribute.....	78
Table 91 – GRID attributes	79
Table 92 – VECTORS attribute	79
Table 93 – ORIENTATION attribute	80
Table 94 – IMAGE attributes.....	80
Table 95 – PATH attribute.....	80
Table 96 – LINK attribute	81
Table 97 – Importing Device Description.....	82
Table 98 – Redefinition attributes	83
Table 99 – Redefinition rules for AXIS attributes.....	83
Table 100 – Redefinition rules for BLOCK_A attributes.....	84
Table 101 – Redefinition rules for BLOCK_B attributes.....	85
Table 102 – Redefinition rules for CHART attributes.....	85
Table 103 – Redefinition rules for COLLECTION attributes.....	86
Table 104 – Redefinition rules for COMMAND attributes.....	86
Table 105 – Redefinition rules for COMPONENT attributes.....	86
Table 106 – Redefinition rules for COMPONENT_FOLDER attributes.....	87
Table 107 – Redefinition rules for COMPONENT_REFERENCE attributes.....	87
Table 108 – Redefinition rules for COMPONENT_RELATION attributes.....	88
Table 109 – Redefinition rules for CONNECTION attributes.....	88
Table 110 – Redefinition rules for DOMAIN attributes	88
Table 111 – Redefinition rules for EDIT_DISPLAY attributes.....	88
Table 112 – Redefinition rules for FILE attributes	89
Table 113 – Redefinition rules for GRAPH attributes.....	89
Table 114 – Redefinition rules for GRID attributes	89
Table 115 – Redefinition rules for IMAGE attributes.....	90
Table 116 – Redefinition rules for INTERFACE attributes	90
Table 117 – Redefinition rules for LIST attributes	90
Table 118 – Redefinition rules for MENU attributes.....	91
Table 119 – Redefinition rules for METHOD attributes	91
Table 120 – Redefinition rules for PROGRAM attributes	91
Table 121 – Redefinition rules for RECORD attributes	92
Table 122 – Redefinition rules for REFERENCE_ARRAY attributes	92
Table 123 – Redefinition rules for RESPONSE_CODES attributes.....	92

Table 124 – Redefinition rules for SOURCE attributes	93
Table 125 – Redefinition rules for TEMPLATE attributes	93
Table 126 – Redefinition rules for VALUE_ARRAY attributes	93
Table 127 – Redefinition rules for VARIABLE attributes	94
Table 128 – Redefinition rules for VARIABLE_LIST attributes	94
Table 129 – Redefinition rules for WAVEFORM attributes	95
Table 130 – INTERFACE attribute	96
Table 131 – DECLARATION attribute	96
Table 132 – LIKE attributes	96
Table 133 – LIST attributes	97
Table 134 – TYPE attribute	97
Table 135 – CAPACITY, COUNT attribute	97
Table 136 – MENU attribute	98
Table 137 – ITEMS attribute	99
Table 138 – ACCESS attribute	99
Table 139 – POST_EDIT_ACTIONS, PRE_EDIT_ACTIONS, POST_READ_ACTIONS, PRE_READ_ACTIONS, POST_WRITE_ACTIONS, PRE_WRITE_ACTIONS attributes	100
Table 140 – PURPOSE attribute	102
Table 141 – ROLE attribute	102
Table 142 – STYLE attribute	103
Table 143 – METHOD attributes	105
Table 144 – ACCESS attributes	105
Table 145 – TYPE attributes	106
Table 146 – PROGRAM attributes	106
Table 147 – ARGUMENTS attribute	107
Table 148 – RECORD attributes	107
Table 149 – REFERENCE_ARRAY attribute	107
Table 150 – ELEMENTS attribute	108
Table 151 – REFRESH attributes	108
Table 152 – UNIT attributes	109
Table 153 – WRITE_AS_ONE attribute	109
Table 154 – RESPONSE_CODES attributes	109
Table 155 – SOURCE attributes	110
Table 156 – Y_AXIS attribute	111
Table 157 – TEMPLATE attributes	112
Table 158 – DEFAULT_VALUES attribute	112
Table 159 – VALUE_ARRAY attributes	112
Table 160 – NUMBER_OF_ELEMENTS attribute	113
Table 161 – TYPE attribute	113
Table 162 – VARIABLE attributes	113
Table 163 – CLASS attributes	114
Table 164 – TYPE attributes	115
Table 165 – DOUBLE, FLOAT, INTEGER, UNSIGNED_INTEGER attributes	117

Table 166 – DATE, DATE_AND_TIME, DURATION, TIME, TIME_VALUE attributes	119
Table 167 – BIT_ENUMERATED attributes	120
Table 168 – status–class attributes	121
Table 169 – ALL, AO, DV, TV attributes	122
Table 170 – Enumerated types attributes	122
Table 171 – Index type attributes	123
Table 172 – Object reference type attribute	123
Table 173 – DEFAULT_REFERENCE attributes	123
Table 174 – String types attributes	125
Table 175 – CONSTANT_UNIT attribute	125
Table 176 – DEFAULT_VALUE attribute	126
Table 177 – INITIAL_VALUE attribute	126
Table 178 – POST_EDIT_ACTIONS, PRE_EDIT_ACTIONS, POST_READ_ACTIONS, PRE_READ_ACTIONS, POST_WRITE_ACTIONS, PRE_WRITE_ACTIONS, REFRESH_ACTIONS attributes	126
Table 179 – READ/WRITE_TIMEOUT attributes	128
Table 180 – STYLE attribute	129
Table 181 – VARIABLE_LIST attributes	129
Table 182 – WAVEFORM attributes	129
Table 183 – TYPE attribute	130
Table 184 – XY attribute	130
Table 185 – YT attribute	131
Table 186 – HORIZONTAL attribute	132
Table 187 – VERTICAL attribute	132
Table 188 – EXIT_ACTIONS, INIT_ACTIONS, REFRESH_ACTIONS attribute	133
Table 189 – KEY_POINTS attribute	133
Table 190 – X_VALUES, Y_VALUES attribute	134
Table 191 – Y_AXIS attribute	134
Table 192 – CLASSIFICATION attribute	135
Table 193 – COMPONENT_PARENT attribute	136
Table 194 – COMPONENT_PATH attribute	136
Table 195 – DEFINITION attributes	137
Table 196 – EMPHASIS attribute	137
Table 197 – HANDLING attribute	138
Table 198 – HEIGHT/WIDTH attribute	138
Table 199 – HELP attribute	138
Table 200 – LABEL attribute	139
Table 201 – LINE_COLOR attribute	139
Table 202 – LINE_TYPE attribute	139
Table 203 – MEMBERS attributes	140
Table 204 – PROTOCOL attribute	141
Table 205 – RESPONSE_CODES attribute	141
Table 206 – SUPPLIED_INTERFACE attribute	141
Table 207 – VALIDITY attributes	142

Table 208 – IF, SELECT conditional	143
Table 209 – Referencing an EDD instance	143
Table 210 – Referencing elements of VARIABLE	144
Table 211 – Referencing elements of RECORD	144
Table 212 – Referencing elements of VALUE_ARRAY	144
Table 213 – Referencing members of COLLECTION	145
Table 214 – Referencing members of REFERENCE_ARRAY	145
Table 215 – Referencing members of VARIABLE_LISTS	145
Table 216 – Referencing members of a BLOCK_A PARAMETERS	145
Table 217 – Referencing members of BLOCK_A PARAMETER_LISTS	146
Table 218 – Referencing members of BLOCK_A LOCAL_PARAMETER	146
Table 219 – Referencing BLOCK_A CHARACTERISTICS	146
Table 220 – Referencing members of FILE	147
Table 221 – Referencing elements of LIST	147
Table 222 – Referencing members of CHART	147
Table 223 – Referencing members of GRAPH	147
Table 224 – Referencing members of SOURCE	148
Table 225 – Referencing AXIS of a GRAPH, SOURCE, WAVEFORM	148
Table 226 – Referencing PARAMETERS of specific BLOCK_A instance	148
Table 227 – Referencing LOCAL_PARAMETERS of specific BLOCK_A instance	149
Table 228 – Referencing CHARACTERISTICS of specific BLOCK_A instance	149
Table 229 – Referencing CHARTS of specific BLOCK_A instance	149
Table 230 – Referencing LISTS of specific BLOCK_A instance	150
Table 231 – Referencing GRAPHS of specific BLOCK_A instance	150
Table 232 – Referencing GRIDS of specific BLOCK_A instance	150
Table 233 – Referencing MENUS of specific BLOCK_A instance	151
Table 234 – Referencing METHODS of specific BLOCK_A instance	151
Table 235 – Referencing a COMPONENT instance	151
Table 236 – Referencing a COMPONENT type	152
Table 237 – string as a string literal	152
Table 238 – string as a string variable	152
Table 239 – String as an enumeration value	153
Table 240 – String as a dictionary reference	153
Table 241 – Referencing HELP and LABEL attributes of EDD instances	153
Table 242 – String operation	154
Table 243 – Format specifier	154
Table 244 – Primary expressions	154
Table 245 – Attribute values of VARIABLES	155
Table 246 – AXIS Attribute Values	155
Table 247 – LIST Attribute Values	156
Table 248 – Unary expressions	156
Table 249 – Multiplicative operators	157
Table 250 – Additive operators	157

Table 251 – Shift operators.....	157
Table 252 – Relational operators	157
Table 253 – Equality operators	158
Table 254 – Text dictionary attributes	159
Table A.1 – Conventions for integer constants	163
Table A.2 – Using escape sequences in string literals.....	164
Table A.3 – Using language codes in string literals	165
Table A.4 – EDDL operators	165
Table A.5 – EDDL keywords	166

IECNORM.COM: Click to view the full PDF of IEC 61804-3:2010

INTERNATIONAL ELECTROTECHNICAL COMMISSION

FUNCTION BLOCKS (FB) FOR PROCESS CONTROL –**Part 3: Electronic Device Description Language (EDDL)**

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 61804-3 has been prepared by subcommittee 65E: Devices and integration in enterprise systems, of IEC technical committee 65: Industrial process measurement, control and automation.

This second edition cancels and replaces the first edition published in 2006. This edition constitutes a technical revision.

The main changes with respect to the previous edition are listed below¹.

Additions

- Language elements to support modular devices, see , 7.9, 7.10, 7.36.1, 7.36.2, 7.36.3, 7.36.13, and 7.36.15;
- Language elements to support offline configuration, 7.31;
- Usage of UTF-8, see A.2.3.;

¹ The historical background to the EDDL specification is given in Annex E.

- Various BuiltIns;
 - BLOCK_A referencing. Updates to support access to multiple blocks, see 7.4.1, 7.38.18 up to 7.38.26.;
 - Add VALIDITY attribute to various lexical structures;
 - Support of multi-language images, see 7.18.2.1.;
 - In 7.23.1 deleted the restriction of the MENU item list;
 - Syntactical limitation on conditionals in 7.23.2.1 to restrict the MENU layout;
 - Add LIST and delete VARIABLE_LIST of the MENU items in 7.23.2.1;
 - Additional return value data types for METHOD TYPE, see 7.24.2.3;
 - Replace reference by a context specific specification in Table 170;
 - Clarification on KEY_POINTS behaviour, see 7.35.2.4;
 - Add TRANSPARENT to the lexical structure to make the list of attributes consistent, see 7.36.11;
 - Clarification on file behaviour, see 7.36.10;
 - Add in Table A.5 the new key-words;
 - Add the formal EDDL syntax in A.6.
- Corrections
 - Deleted in A.6 all non-needed constructs using the auxiliary ..._listR. These were created by a non-perfect syntax-checking tool.
 - Deleted in A.6 all non-needed "stmt1:" and "stmt2:". These were created by a non-perfect syntax-checking tool.
 - Made A.6 consistent about using a colon at the end of a term by amending colons in a consistent way.
 - Deleted not used references.
 - Deleted in several lexical structures the brackets and "<exp>"
 - Deleted in lexical structures the "[... <expr>]" where the conditional expression was not possible.
 - Exchanged the attribute WIDTH by HEIGHT in the subclause specifying HEIGHT.
 - Spelling errors like GUAGE exchanged to GAUGE.
 - Syntactical limitation on conditionals in ACTIONS (for example see 7.14.2.3 and 7.14.2.4) to support conditionals only in the METHODS.
 - Clarification on file behavior in 7.15.
 - Add GRID and IMAGES to the attribute list, see 7.19.
 - Deleted a duplication of element list in 7.23.2.1.
 - Defining and calling METHODS with parameter and return value, see 7.24.
 - Restriction on METHOD CLASS, see 7.24.2.2.
 - Clarification on SCALING_FACTOR behaviour, see Table 165.
 - Deleted EDDL operators from EDDL keyword list, see Table A.5.
 - Clarification on TIME_VALUE coding providing the absolute basis, see Table D.16 and Table D.17.

This bilingual version, published in 2011-07, corresponds to the English version.

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

FDIS	Report on voting
65E/162/FDIS	65E/173/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.

The French version of this standard has not been voted upon.

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

A list of all parts of the IEC 61804 series, published under the general title *Function blocks (FB) for process control*, can be found on the IEC web site.

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

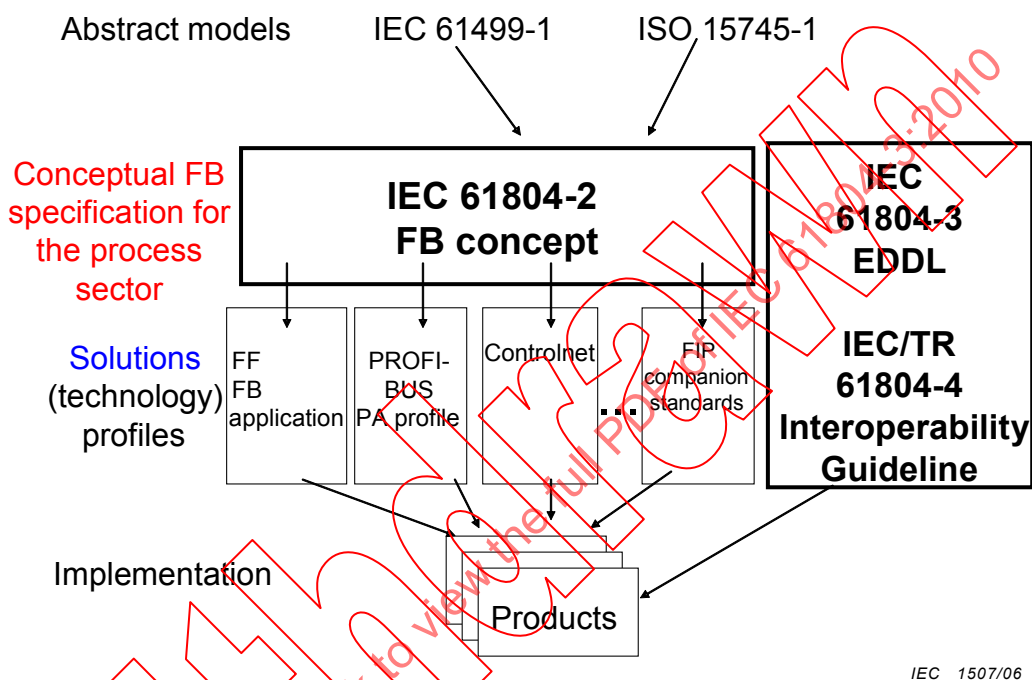
- reconfirmed,
- withdrawn,
- replaced by a revised edition, or
- amended.

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 the IEC 61804 series specifies the Electronic Device Description Language (EDDL) technology.

The EDDL fills the gap between the conceptual Function Block specification of IEC 61804-2 and a product implementation. It allows the manufacturers to use the same description method for devices based on different technologies and platforms. Figure 1 shows these aspects.



IEC 1507/06

Figure 1 – Position of the IEC 61804 series related to other standards and products

FUNCTION BLOCKS (FB) FOR PROCESS CONTROL –

Part 3: Electronic Device Description Language (EDDL)

1 Scope

This part of IEC 61804 specifies the Electronic Device Description Language (EDDL) technology, which enables the integration of real product details using the tools of the engineering life cycle.

This standard specifies EDDL as a generic language for describing the properties of automation system components. EDDL is capable of describing

- device parameters and their dependencies;
- device functions, for example, simulation mode, calibration;
- graphical representations, for example, menus;
- interactions with control devices;
- graphical representations:
 - enhanced user interface;
 - graphing system;
- persistent data store.

EDDL is to be used to create Electronic Device Description (EDD). This EDD is used with appropriate tools to generate interpretative code to support parameter handling, operation, and monitoring of automation system components such as remote I/Os, controllers, sensors, and programmable controllers. Tool implementation is outside the scope of this standard.

This standard specifies the semantic and lexical structure in a syntax-independent manner. A specific syntax is defined in Annex A, but it is possible to use the semantic model also with different syntaxes.

NOTE 1 The EDDL may also be used for the description of product properties in other domains.

The EDDL and the device-related EDD is applicable to industrial automation.

NOTE 2 Industrial automation may include devices such as generic digital and analog input/output modules, motion controllers, human machine interfaces, sensors, closed-loop controllers, encoders, hydraulic valves, and programmable controllers.

This International Standard satisfies the requirements of Clause 9 of IEC/TS 61804-1.

2 Normative references

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

IEC 61499-1:2005, *Function blocks – Part 1: Architecture*

IEC/TS 61804-1:2003, *Function blocks (FB) for process control – Part 1: Overview of system aspects*

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

ISO/IEC 2375:2003, *Information technology – Procedure for registration of escape sequences and coded character sets*

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

ISO/IEC 8859-1:1998, *Information technology – 8-bit single-byte coded graphic character sets – Part 1: Latin alphabet No. 1*

ISO/IEC 9899, *Programming languages – C*

ISO/IEC 10646-1:2000, *Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Part 1: Architecture and Basic Multilingual Plane*

ISO 639, *Code for the representation of names of languages*

ISO 3166-1:2006, *Codes for the representation of names of countries and their subdivisions – Part 1: Country codes*

IEEE 754:1985 (R1990), *Binary Floating-Point Arithmetic*

RFC 3629:2003, *UTF-8, User Datagram Protocol*, available at
<<http://www.ietf.org/rfc/rfc0768.txt>>

3 Terms, definitions, abbreviated terms and acronyms

3.1 Terms and definitions

For the purposes of this document, the following terms and definitions, some of which have been compiled from the referenced documents, apply.

NOTE The terms in IEC 60050-351 apply partially.

3.1.1

algorithm

finite set of well-defined rules for the solution of a problem in a finite number of operations

3.1.2

attribute

description of an externally visible characteristic or feature of an object

NOTE The attributes of an object contain information about variable portions of an object. Typically, they provide status information or govern the operation of an object. Attributes may also affect the behaviour of an object. Attributes are divided into class attributes and instance attributes.

3.1.3

Builtin

predefined subroutine for communication and display executed by the EDD application

3.1.4

configuration (of a system or device)

step in system design: selecting functional units, assigning their locations and defining their interconnections

[IEC 61499-1]

3.1.5

data

representation of facts, concepts or instructions in a formalized manner suitable for communication, interpretation or processing by human beings or by automatic means

[ISO/AFNOR Dictionary of Computer Science]

3.1.6

data connection

association established between functional units for conveyance of data

[IEC 61499-1]

3.1.7**data input**

interface of an FB which receives data from a data connection

[IEC 61499-1]

3.1.8**data output**

interface of an FB, which supplies data to a data connection

[IEC 61499-1]

3.1.9**data type**

set of values together with a set of permitted operations

[ISO 2382 series]

3.1.10**device**

independent physical entity capable of performing one or more specified functions in a particular context and delimited by its interfaces

[IEC 61499-1]

3.1.11**EDD application**

program using the EDD, or any translated form, which offers functionality like communication representation, data representation, graphical representation, etc.

3.1.12**EDDL processor**

processor or program, which translates the EDD into an executable form that can be processed by an EDD application

3.1.13**EDDL profile**

selection of the supported elements of the EDDL lexical structure including the syntax definitions for a number of specific consortia

3.1.14**Electronic Device Description Language****EDDL**

methodology for describing parameter(s) of an automation system component

3.1.15**Electronic Device Description****EDD**

data collection containing the device parameter(s), their dependencies, their graphical representation and a description of the data sets which are transferred.

NOTE The Electronic Device Description is created using the Electronic Device Description Language (EDDL).

3.1.16**Electronic Device Description Source****EDDS**

ASCII file containing a specific device description

3.1.17

Electronic Device Description Technology

EDDT

technology which includes the EDD development process, the EDD usage and the involved tool chain

3.1.18

Electronic Device Description Checker

test tool which checks the Lexical Structure and partly the semantic of EDD sources to guarantee compliance of EDD sources with the EDD language

3.1.19

Electronic Device Description Compiler

compiler which translates the EDD source in an internal format that is used by the EDD Interpreter

3.1.20

Electronic Device Description Interpreter

EDDI

Interpreter which uses the EDD source or an internal format that is given by the EDDL Compiler to provide the EDD information to the EDD user

3.1.21

entity

particular thing, such as a person, place, process, object, concept, association, or event

[IEC 61499-1]

3.1.22

event

instantaneous occurrence that is significant to scheduling the execution of an algorithm

[IEC 61499-1]

NOTE The execution of an algorithm may make use of variables associated with an event.

3.1.23

exception

event that causes suspension of normal execution

[IEC 61499-1]

3.1.24

function

specific purpose of an entity or its characteristic action

[IEC 61499-1]

3.1.25

functional unit

entity of hardware or software, or both, capable of accomplishing a specified purpose

[ISO/AFNOR Dictionary of Computer Science]

3.1.26

function block (function block instance)

software functional unit comprising an individual, named copy of a data structure and associated operations specified by a corresponding FB type

[IEC 61499-1]

NOTE Typical operations of a FB include modification of the values of the data in its associated data structure.

3.1.27**hardware**

physical equipment, as opposed to programs, procedures, rules and associated documentation

[ISO/AFNOR Dictionary of Computer Science]

3.1.28**implementation**

development phase in which the hardware and software of a system become operational

[IEC 61499-1]

3.1.29**input variable**

variable whose value is supplied by a data input, and which may be used in one or more operations of a FB

NOTE An input parameter of an FB, as defined in IEC 61131-3, is an input variable.

[IEC 61499-1]

3.1.30**instance**

functional unit comprising an individual, named entity with the attributes of a defined type

[IEC 61499-1]

3.1.31**instance name**

identifier associated with, and designating, an instance

[IEC 61499-1]

3.1.32**instantiation**

creation of an instance of a specified type

[IEC 61499-1]

3.1.33**interface**

shared boundary between two functional units, defined by functional characteristics, signal characteristics, or other characteristics as appropriate

[IEC 60050-351:1998, 351-11-19, modified]

3.1.34**internal variable**

variable whose value is used or modified by one or more operations of an FB but is not supplied by a data input or to a data output

[IEC 61499-1]

3.1.35**invocation**

process of initiating the execution of the sequence of operations specified in an algorithm

[IEC 61499-1]

3.1.36

mapping

set of values having defined correspondence with the quantities or values of another set

[ISO/AFNOR Dictionary of Computer Science]

3.1.37

model

representation of a real world process, device, or concept

[IEC 61499-1]

3.1.38

operation

well-defined action that, when applied to any permissible combination of known entities, produces a new entity

[ISO/AFNOR Dictionary of Computer Science]

3.1.39

output variable

variable whose value is established by one or more operations of a FB, and is supplied to a data output

NOTE An output parameter of an FB, as defined in IEC 61131-3, is an output variable.

[IEC 61499-1]

3.1.40

parameter

variable that is given a constant value for a specified application and that may denote the application

[ISO/AFNOR Dictionary of Computer Science]

3.1.41

preprocessor

part of the run-time environment that transforms the information given in an EDD

3.1.42

preprocessor directives

description of conditions for filtering the EDD code before compilation or interpretation

NOTE For example, a preprocessor directive providing the facility to define names for constants or to write macros to make code easier to read.

3.1.43

resource

functional unit contained within a device which has independent control of its operation, and which provides various services to applications, including the scheduling and execution of algorithms

NOTE 1 The RESOURCE defined in IEC 61131-3 is a programming language element corresponding to the resource defined above.

NOTE 2 A device contains one or more resources.

3.1.44

resource management application

application whose primary function is the management of a single resource

[IEC 61499-1]

3.1.45**service**

functional capability of a resource, which can be modelled by a sequence of service primitives

[IEC 61499-1]

3.1.46**software**

intellectual creation comprising the programs, procedures, rules and any associated documentation pertaining to the operation of a system

[IEC 61499-1]

3.1.47**system**

set of interrelated elements considered in a defined context as a whole and separated from its environment

[IEC 60050-351:IEV 351-21-20:2006, modified]

NOTE 1 Such elements may be both material objects and concepts as well as the results thereof (for example, forms of organization, mathematical methods, and programming languages).

NOTE 2 The system is considered to be separated from the environment and other external systems by an imaginary surface, which can cut the links between them and the considered system.

3.1.48**text dictionary**

collection of multilingual or other texts within the EDD

NOTE References within an EDD are used to select an appropriate text dictionary.

3.1.49**type**

software element, which specifies the common attributes shared by all instances of the type

[IEC 61499-1]

3.1.50**type name**

identifier associated with, and designating, a type

[IEC 61499-1]

3.1.51**variable**

software entity that may take different values, one at a time

NOTE 1 The values of a variable are usually restricted to a certain data type.

NOTE 2 Variables are described as input variables, output variables, and internal variables.

[IEC 61499-1]

3.2 Abbreviated terms and acronyms

ANSI	American National Standard Institute
ANSI C	American National Standard Institute for the programming language C (see ISO/IEC 9899)
ASCII	American Standard Code for Information Interchange (see ISO/IEC 10646-1)
ASN.1	Abstract Lexical Structure Notation 1

CP	Communication Profile
CPF	Communication Profile Family
DD	Device Description
EDD	Electronic Device Description
EDDI	Electronic Device Description Interpreter
EDDL	Electronic Device Description Language
EUC	Extended Unit Code (see ISO/IEC 2022)
FB	Function Block
FIFO	First In First Out
HCF	HART® ² Communication Foundation
HMI	Human Machine Interface
HTML	Hypertext Mark-up Language
I/O	Input/Output
ID	Identifier
OSI	Open Systems Interconnection
PI	PROFIBUS & PROFINET ³ International
PNO	PROFIBUS Nutzerorganisation
wao	Write as one

4 Conformance statement

A statement of conformance to a profile of this International Standard shall be stated⁴ as either

- compliance to IEC 61804-3:2010 Annex D.n <Type>

where the Type within the angle brackets < > is optional and the angle brackets are not to be included.

NOTE The Type may be, for example, the name of the communication protocol used like PROFIBUS, Foundation Fieldbus, or HART.

Product standards shall not include any conformity assessment aspects (including QM provisions), neither normative nor informative, other than provisions for product testing (evaluation and examination).

5 Conventions for lexical structures

The EDDL is generally described using lexical structures in which the elements and the presence of fields are specified. A general form of the lexical structures is shown below.

² HART® is a registered trademark of the consortium HART Communication Foundation (HCF) (non-profit organization). This information is given for the convenience of users of this standard and does not constitute an endorsement by IEC of the trade name holder or any of its products. Compliance to this profile does not require use of the trade name. Use of the trade names requires permission from the trade name holder.

³ PROFIBUS & PROFINET are trade names of PROFIBUS Nutzerorganisation e.V. (PNO) as part of PROFIBUS & PROFINET International. PNO is a non-profit trade organization to support the fieldbus technologies PROFIBUS & PROFINET. This information is given for the convenience of users of this standard and does not constitute an endorsement by IEC of the trade name holder or any of its products. Compliance to this profile does not require use of the tradename. Use of the trade names requires permission from the trade name holder.

⁴ In accordance with the ISO/IEC Directives.

ABC field1, field2

ABC is a lexical element. This element shall be coded in a concrete syntax. It is not required to code this element with the name “ABC”. It is also possible to code this element, for example, with a tag number.

Field1 and field2 are fields of the lexical element ABC. Each field is mandatory and may have more than one attribute. If a field has attributes, the presence of the attributes is specified in a table. A comma separates Field1 and 2. The comma is a lexical element and is not coded explicitly.

If a field has additional attributes, the attributes are defined in a table. The table layout and the possible usage qualifiers are shown in Table 1.

Table 1 – Field attribute descriptions

Usage	Attribute	Description
m	yyy	The presence of this attribute is mandatory.
o	xxx	The presence of this attribute is optional.
s	z1	The presence of this attribute is selectable with other attributes, which are also marked with “s” in the usage column). Only one of the selectable attributes (z1 or z2) is present.
s	z2	The presence of this attribute is selectable with other attributes, which are also marked with “s” in the usage column). Only one of the selectable attributes (z1 or z2) is present.
c	uuu	The presence of this attribute is conditional and it is only present if the condition is true.

The characters in the Usage column have the following meanings.

- m: this attribute is mandatory and shall be present.
- o: this attribute is optional and need not be present.
- s: this attribute is a selection. One, and only one, of the fields marked with s (Z1 or Z2) shall be present.
- c: this attribute is conditional. The condition is described in the Description column.

In the Attribute column, if more than one attribute exists, which have the same usage, the attributes are sorted in alphabetical order.

ABC field1+

The plus (+) behind field1 is used to indicate that field1 is used at least once. It may be used more than once.

ABC field2*

The asterisk (*) behind field2 is used to indicate that field2 is optional and need not be used. If it is used it may be used more than once.

ABC [field1, field2]+

The elements field1 and field2 in the square brackets [] are an unsorted list. The plus behind the closing bracket indicates that field1 and field2 are used at least once and may be used more than once as group.

ABC field1, (field2, field3)<exp>

<exp> indicates that field2 and field3 are used in conjunction with the conditional expression, (conditional constructs are specified in 7.37). The expression <exp> refers only to the fields within the preceding curved brackets (). Usage of conditional expressions is optional.

6 EDD and EDDL model

6.1 Overview of EDD and EDDL

An Electronic Device Description (EDD) contains all device parameter(s) of an automation system component. The technology used to describe an EDD is called Electronic Device Description Language (EDDL). The EDDL provides a set of scalable basic language elements to handle simple, complex or modular devices. The EDDL is a descriptive language based on an ASCII format with clear separation between data and program.

NOTE Data in a text field, which is marked with a country code like Japanese, may use multi-byte code.

6.2 EDD architecture

From the viewpoint of the ISO/OSI model (ISO/IEC 7498-1), an EDD is above Layer 7. However, the EDD application uses the communication system to transfer its information. An EDD contains constructs that support mapping to a supporting communication system.

The device manufacturer defines the objects, which are reflected by the logical representation of the objects within an EDD application. For that reason, EDDL has language elements, which map the EDD data to the data representation of the communication system, so that the user of an EDD does not need to know the physical location (address) of a device object.

EDD describes the management of information to be displayed to the user. The specific representation of such visualization is not part of an EDD or EDDL definitions.

6.3 Concepts of EDD

The manufacturer of a device or of an automation system component describes the properties of the device by using the EDDL. The resulting EDD contains information such as:

- description of the device parameters;
- description of parameter dependencies;
- logical grouping of the device parameters;
- selection and execution of supported device functions;
- description of the transferred data sets.

Depending on the required usage, the EDD may be physically located

- in a device;
- in an external data storage medium such as a compact disk, floppy or a server;
- partially distributed in the device and an external storage medium.

EDD supports text strings (common terms, phrases etc.) in more than one language (English, German, French, etc.). Text strings may be stored in separate dictionaries. There may be more than one dictionary for one EDD.

An EDD implementation includes sufficient information about the target device, for example, manufacturer, device type, revision, etc. This is used to match a specific EDD to a specific device.

6.4 Principles of the EDD development process

6.4.1 General

The creation of an EDD is a three-stage process: EDD source generation, EDD preprocessing and EDD compilation.

6.4.2 EDD source generation

The EDD generation process is shown in Figure 2.

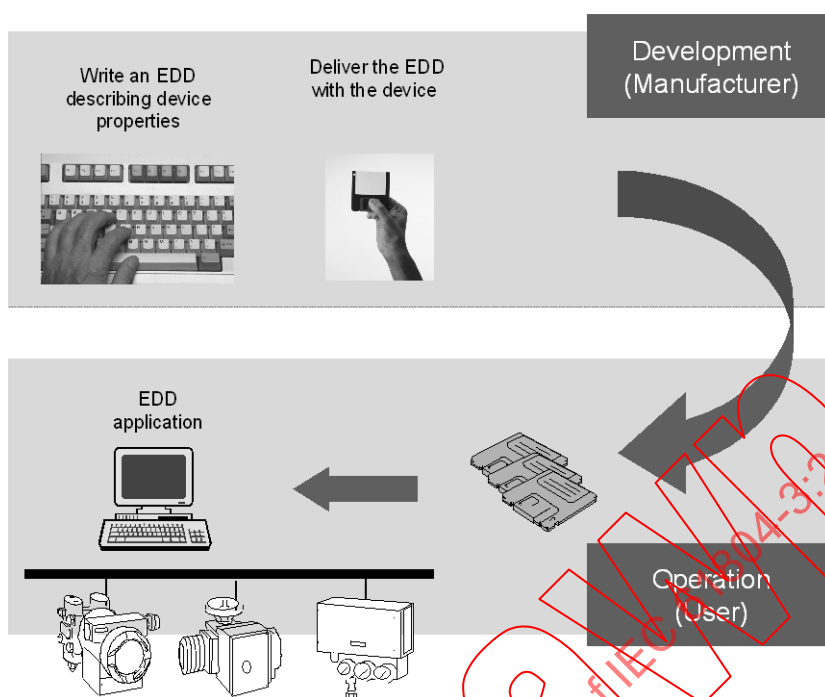


Figure 2 – EDD generation process

IEC 1511/06

The device manufacturer writes an appropriate EDD for his device and delivers both (EDD and device). If the automation system supports the EDD method, a system integration can be made by the user.

The EDDs for devices may be embedded in device memory or delivered using separate storage media or downloaded from an appropriate network server. The EDD is 'interpreted' or 'browsed' by an EDD-application. EDDs are normally stored as source files or preprocessed files.

6.4.3 EDD preprocessing

In the preprocessing stage an EDD preprocessor generates a consistent EDD representation suitable for final compilation.

Preprocessing supports, for example, substitution of definitions and inclusion of external text. The output of the preprocessor is a complete EDD without any preprocessor directive.

6.4.4 EDD compilation

The EDD compilation stage produces an EDD application internal representation from a preprocessed EDD to be used in the EDD application.

6.5 Interrelations between the lexical structure and formal definitions

The lexical structure of EDDL and its elements are described in Clause 7. Formal definitions and syntax for each EDDL element are given in Annex A. The lexical structure and its formal description use the same name.

NOTE Instead of the specified formal definitions and syntax in Annex A, another specification may be developed as an additional option.

6.6 Builtins

Builtins are predefined subroutines which are executed in the EDD application.

EXAMPLE A hand-held terminal is a simple device having a small display and limited cursor functions. For this type of device, a Builtin could be specified to provide display entry using only up/down right/left cursor actions.

The library of Builtins is defined in Annex B.

6.7 Profiles

EDDL is a harmonized specification of existing legacy EDD concepts. Concrete EDD applications use a subset of the EDDL specification. Selection of EDDL elements and Builtins is made from the profiles defined in Annex D.

In addition to EDD profiles, implementing consortia also publish “Device Profiles”, which are used to support interchangeability of compliant devices. These Device Profiles may be described using the EDDL specified in this document.

7 Electronic Device Description Language

7.1 Overview

7.1.1 EDDL features

The EDDL is a structured and interpretative language for describing device properties. Also the interactions between the devices and the EDD run-time environment are incorporated in the EDDL. The EDDL provides a set of language elements for these purposes.

For a specific EDD implementation it is not necessary to use all of the elements provided by the language.

Compatible subsets of EDDL are permitted and may be specified using profiles (for example, choice of constructs, number of recursions, and selection of options). Profiles supported by some industrial consortia are specified in Annex D. EDD developers are required to identify within each device details as to which profile has been used.

7.1.2 Syntax representation

The specification of the EDDL language in this clause of the standard uses an abstract syntax. The abstract lexical structure in this section is converted into specific syntaxes specified in Annex A by the element name (for example, VARIABLE in the lexical structure equates to the keyword “VARIABLE” in the annex).

NOTE Beside the defined syntax in this standard, it is possible to use other syntax definitions, which may be added in future to allow other features and representations for future progress.

7.1.3 EDD language elements

The language is structured in the following language elements:

- identification element;
- basic construction elements;
- special elements.

Frequently used text strings, for example, help text and multi-lingual lists of labels should be separated in a text dictionary (see 7.41).

Identification information shall be the first entry in every EDD file and shall appear only once. The identification information uniquely identifies the version of EDDL used, together with the specific device type, model codes and revision details covered by the EDD file.

7.1.4 Basic construction elements

These basic constructs have been specified to support descriptions of devices used within industrial control applications, together with their properties and functionality.

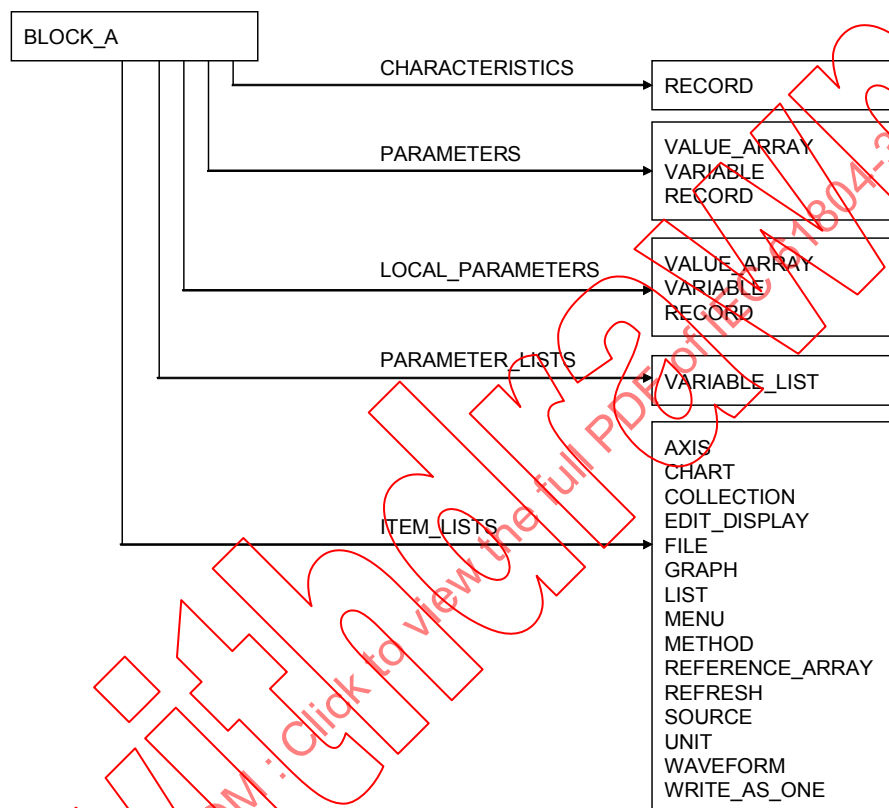
Some constructs have similar names and functions while differing in their detailed specification. This additional variety has been included to ensure compatibility with several existing description languages. Appropriate mapping cross-references are given by profiles.

Each of the basic constructs has a set of attributes associated with it. Attributes can also have subattributes, which refine the definition of the attribute and hence the definition of the construct itself.

The definition of an attribute may be static or dynamic. A static attribute definition never changes, while a dynamic attribute definition may change in order to accommodate parameter value changes.

Some basic constructs refer to other basic constructs. In the following list, the basic constructs as well as the relationships between the basic constructs are shown:

- **AXIS** describes the axis of a CHART or GRAPH (see 7.3).
- **BLOCK_A** is a logical grouping of CHARACTERISTICS, PARAMETERS, PARAMETER_LISTS, and ITEM_LISTS, see Figure 3. To access one item of BLOCK_A, the instance of the block should be used (see 7.4.1 and Figure 3).



IEC 1512/06

Figure 3 – BLOCK_A

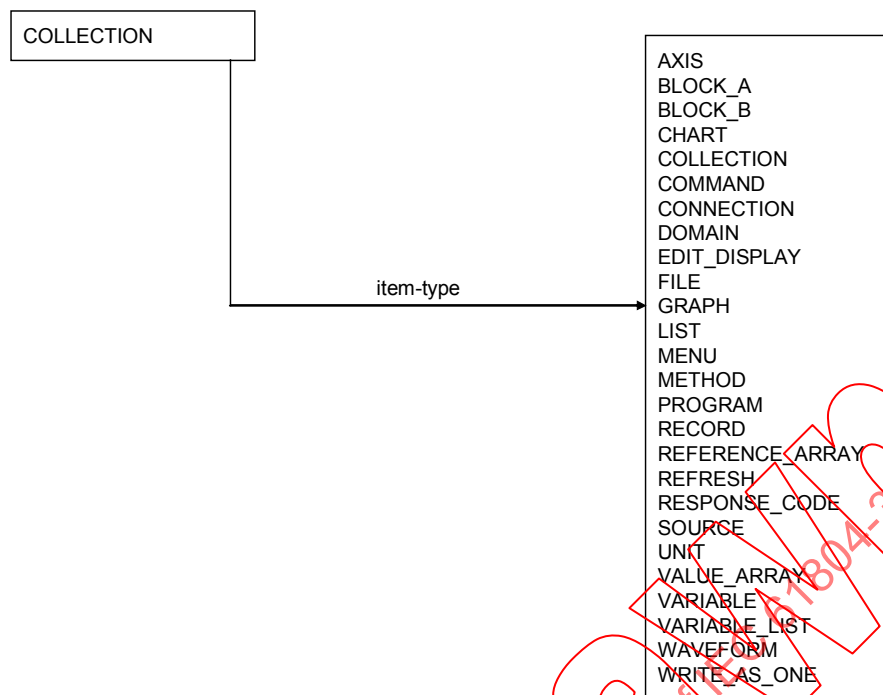
- **BLOCK_B** is used to structure variables in instances of logical block types. For accessing a VARIABLE within a BLOCK_B construct, the basic construct COMMAND is used to provide the relative addressing (see 7.4.2).
- **CHART** describes a chart used to display data from a device in the EDD application (see 7.5 and Figure 4).



IEC 1513/06

Figure 4 – CHART

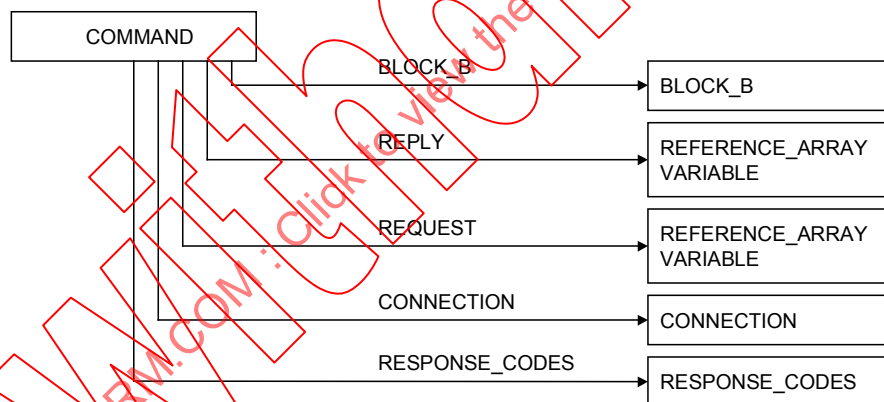
- **COLLECTION** describes logical groupings of data (see 7.6 and Figure 5).



IEC 1514/06

Figure 5 – COLLECTION

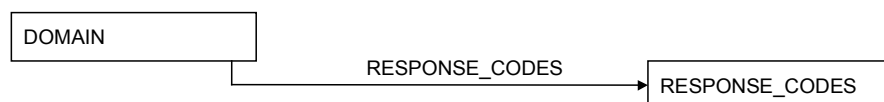
- **COMMAND** describes the structure and the addressing of the variables in the device (see 7.7 and Figure 6).



IEC 1515/06

Figure 6 – COMMAND

- **CONNECTION** defines multiple applications in a device (see 7.8).
- **DOMAIN** can be used to download or upload moderately large amounts of data to or from a device (see 7.13 and Figure 7).



IEC 1516/06

Figure 7 – DOMAIN

- **EDIT_DISPLAY** describes how the data will be represented to a user by a display device (see 7.14 and Figure 8).

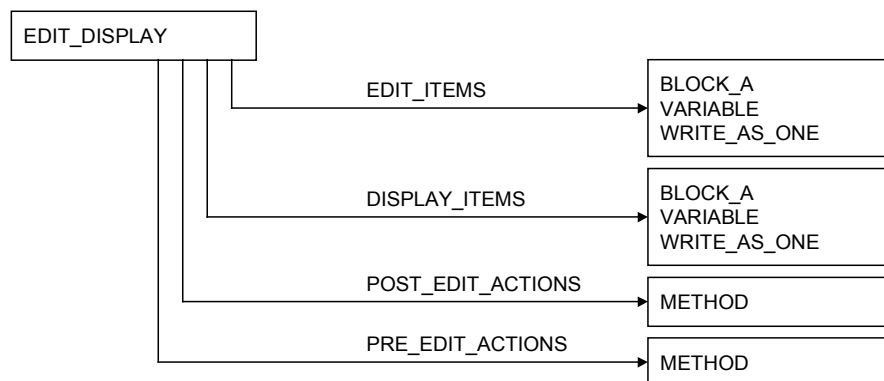


Figure 8 – EDIT_DISPLAY

IEC 1517/06

- **FILE** describes an area of persistent storage (see 7.15 and Figure 9).

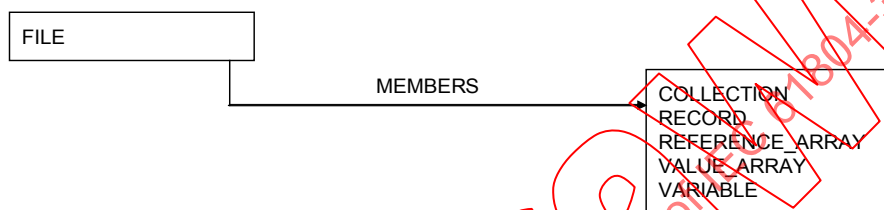


Figure 9 – FILE

IEC 1518/06

- **GRAPH** describes a graph used to display data from a device in the EDD application (see 7.16 and Figure 10).

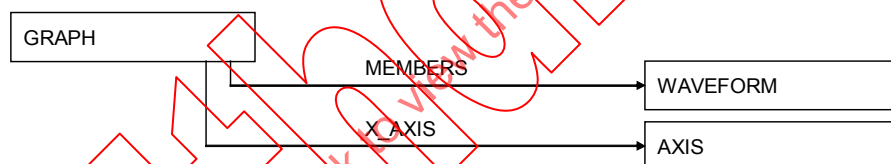
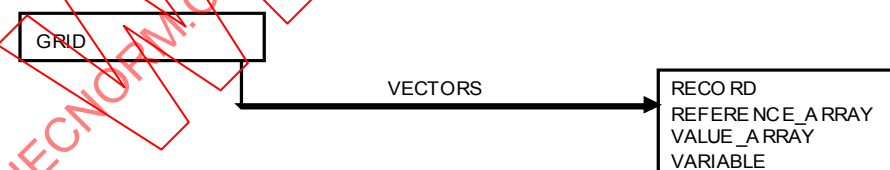


Figure 10 – GRAPH

IEC 1519/06

- **GRID** describes a matrix used to display data from a device in the EDD application (see 7.17 and Figure 11).



IEC 1520/06

Figure 11 – GRID

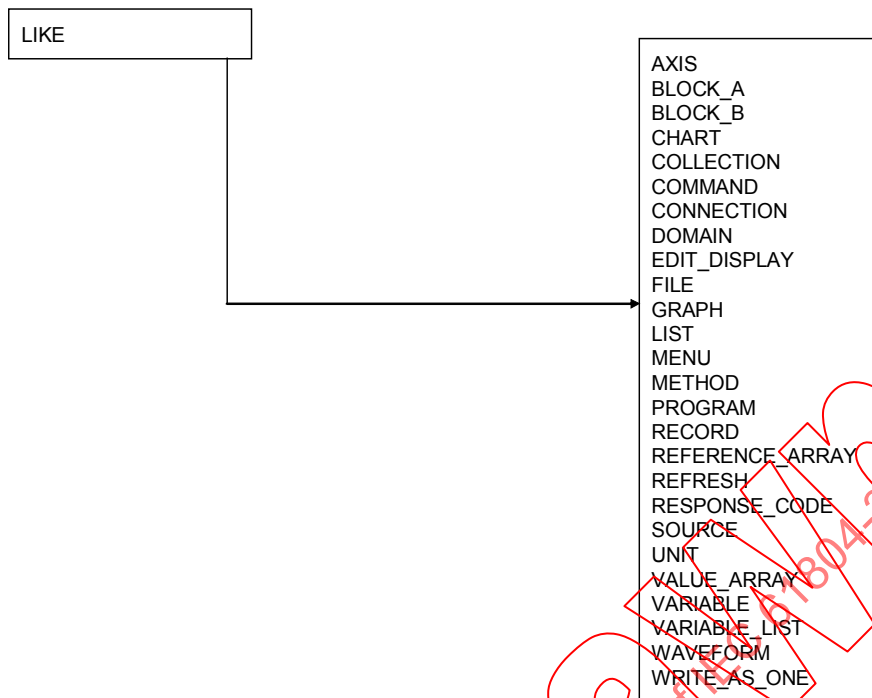
- **IMAGE** describes a visual image (see 7.18 and Figure 12).



Figure 12 – IMAGE

IEC 1521/06

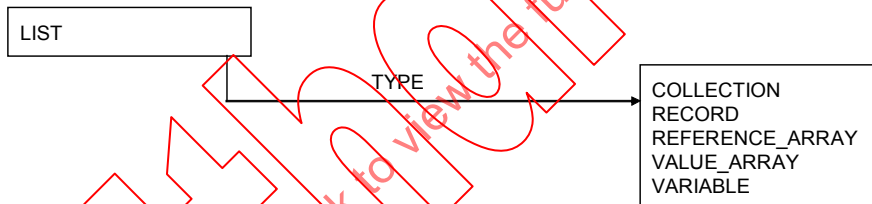
- **IMPORT** is used to import an EDD and to modify its existing definitions (see 7.19).
- **LIKE** creates a new instance of an existing instance of a basic construct. The attributes of the new instance may be redefined, added or deleted (see 7.20 and Figure 13).



IEC 1522/06

Figure 13 – LIKE

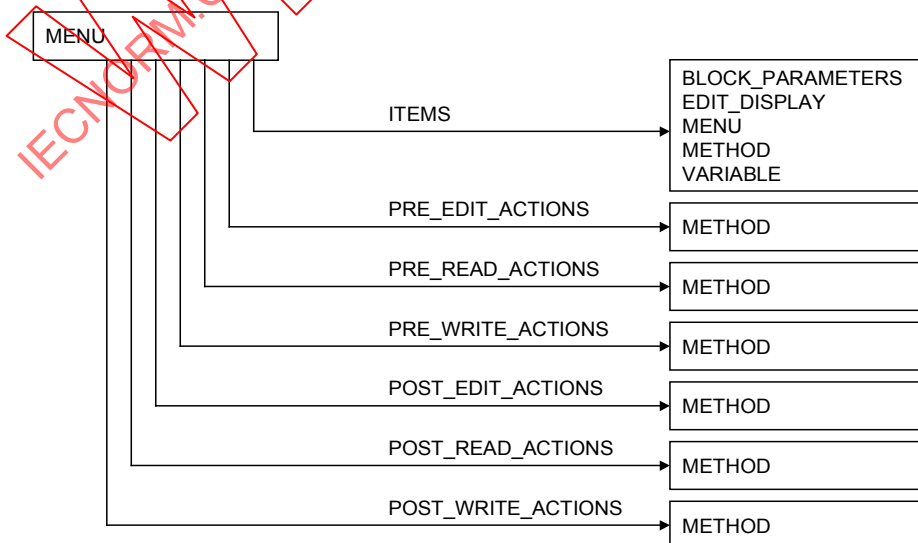
- **LIST** describes a sequence of elements (see 7.22 and Figure 14).



IEC 1523/06

Figure 14 – LIST

- **MENU** describes how the data will be presented to a user by an EDD application (see 7.23 and Figure 15).



IEC 1524/06

Figure 15 – MENU

- **METHOD** describes the execution of complex sequence of event interactions that shall take place between system devices such as display units, configurators and field devices (see 7.24).
- **PROGRAM** specifies how device executable code can be initiated by an appropriate agent (see 7.25 and Figure 16).



Figure 16 – PROGRAM

- **RECORD** describes the data contained in the device (see 7.26 and Figure 17).

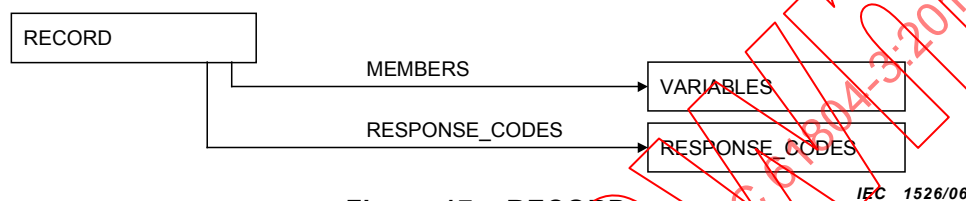


Figure 17 – RECORD

- **REFERENCE_ARRAY** is a set of EDD items of the same EDDL item type (for example, VARIABLE or MENU). An item can be referenced in the EDD via the REFERENCE_ARRAY name and the index associated with the item (see 7.27 and Figure 18).

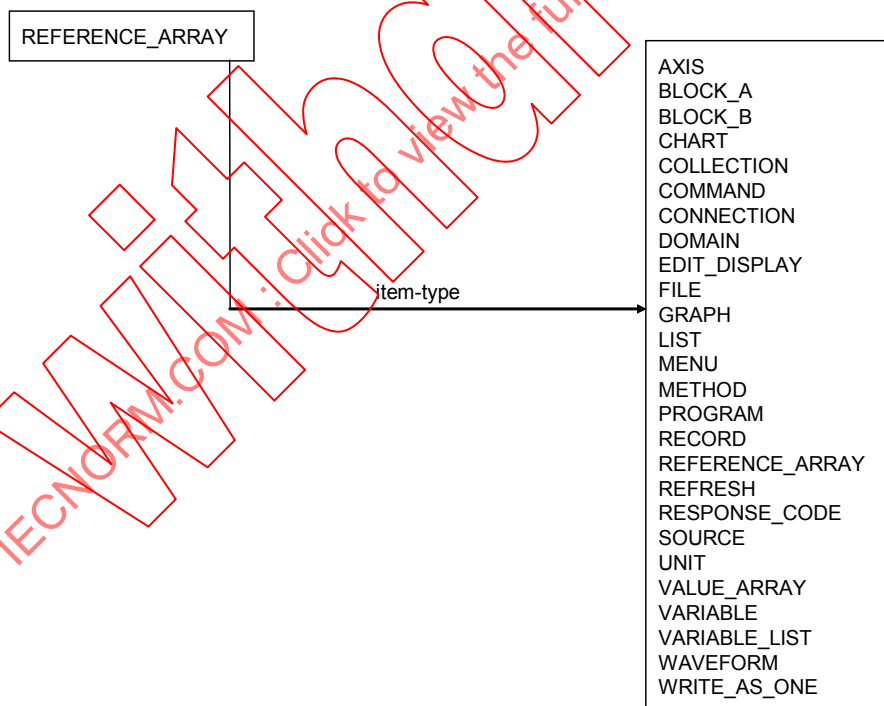


Figure 18 – REFERENCE_ARRAY

- **Relations (see 7.28):** three types of relations are specified.
 - **REFRESH** describes relationships between VARIABLES, RECORDs, and VALUE_ARRAYs (see 7.28.1 and Figure 19).



IEC 1528/06

Figure 19 – REFRESH

— UNIT describes relationships between VARIABLES, RECORDs, and VALUE_ARRAYs (see 7.28.2 and Figure 20).



IEC 1529/06

Figure 20 – UNIT

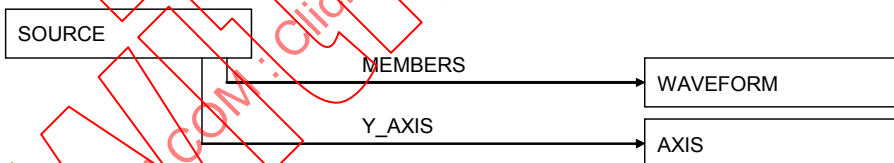
— WRITE_AS_ONE RELATION describes relationships between VARIABLES, RECORDs, and VALUE_ARRAYs (see 7.28.3 and Figure 21).



IEC 1530/06

Figure 21 – WRITE_AS_ONE

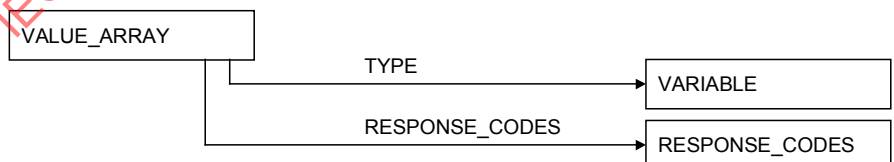
- **RESPONSE_CODES** specifies the application specific response codes for use when managing a VARIABLE, RECORD, VALUE_ARRAY, VARIABLE_LIST, PROGRAM, or DOMAIN (see 7.29).
- **SOURCE** (see 7.30 and Figure 22).



IEC 1531/06

Figure 22 – SOURCE

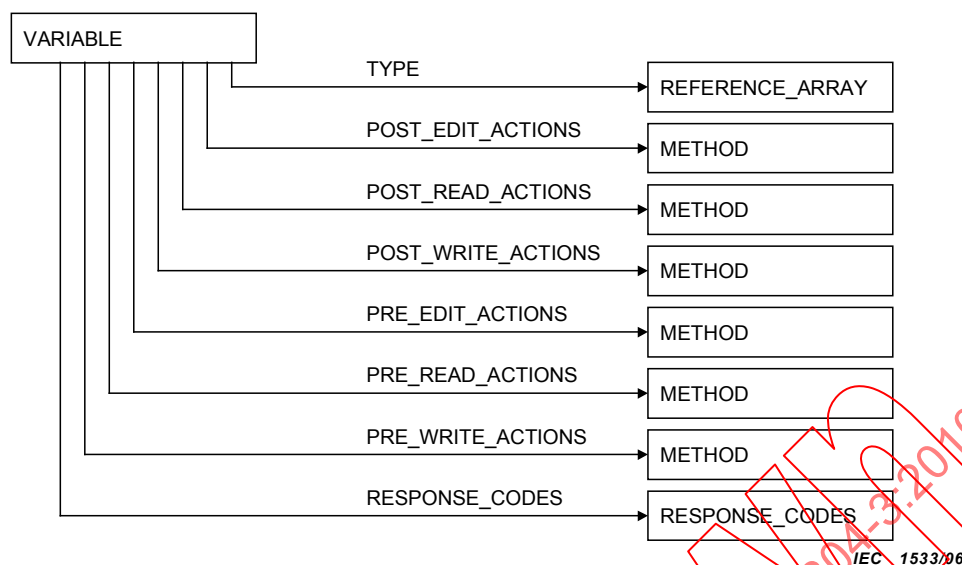
- **VALUE_ARRAY** describes logical groups of EDDL variables (see 7.30.2.4 and Figure 23).



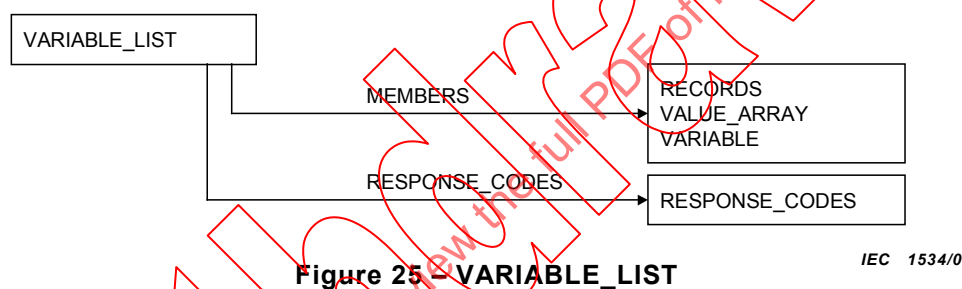
IEC 1532/06

Figure 23 – VALUE_ARRAY

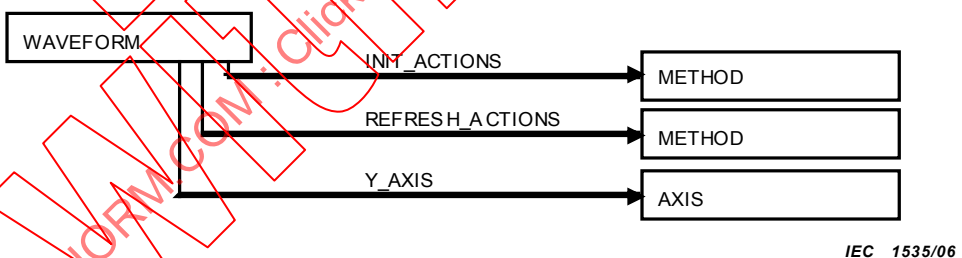
- **VARIABLE** describes the data contained in the device (see 7.33 and Figure 24).



- **VARIABLE_LIST** describes logical groupings of data contained in the device that may be communicated as a list (see 7.34 and Figure 25).



- **WAVEFORM** (see 7.35 and Figure 26).



7.1.5 Common attributes

The following common attributes are used in many basic constructs:

- CLASSIFICATION
- COMPONENT_PARENT
- COMPONENT_PATH
- DEFINITION
- EMPHASIS
- HANDLING
- HEIGHT
- HELP
- LABEL

- LINE_COLOR
- LINE_TYPE
- MEMBERS
- PROTOCOL
- RESPONSE_CODES
- SUPPLIED_INTERFACE
- VALIDITY
- WIDTH

7.1.6 Special elements

The special elements are additional EDDL mechanisms to support file handling, multiple instances and modification of EDDL basic elements, such as:

- **Conditional Expressions** specify attributes values, which are dependent on the values of variables at run-time
- **Reference** is used throughout a device description to refer to other items within the same EDDL
- **Expression** is a logical or mathematical operation on variable values in order to modify attribute values of a basic construct at run-time

7.1.7 Rules for instances

Each instance of a basic construct shall be identified by an identifier of type string. This identifier shall be unique within the complete EDD. All EDDL constructs represent actual instances of device information. If a device contains two or more pieces of information which have the same structure the appropriate construct shall be repeated in the EDD with different identifiers. Alternatively the LIKE construct may be used to create a copy of the construct with a different identifier. The identifier in the lexical definition part is not described within each construct again.

7.1.8 Rules for list of VARIABLES

A list of VARIABLES may contain VARIABLES, elements of composed types or composed types. Composed types are:

- BLOCK_A PARAMETERS
- COLLECTIONS
- RECORDS
- REFERENCE_ARRAY
- VALUE_ARRAY
- VARIABLE_LISTS

Instead of referencing every single member of a composed type, a reference can be made to the type instance.

7.2 EDD identification information

7.2.1 General structure

Purpose

EDD identification information uniquely identifies the device description of a specific device type from a device manufacturer. It also specifies the EDDL version and EDDL profile used.

This identification information shall be the first entry in every EDD and shall appear only once.

Lexical structure

DD_REVISION, DEVICE_REVISION, DEVICE_TYPE, EDD_PROFILE, EDD_VERSION,
MANUFACTURER, MANUFACTURER_EXT

NOTE 1 EDD identification information is not structured in the same way as other EDDL constructs.

NOTE 2 Strict order of the keywords of the EDD identification information is not necessary.

7.2.2 Specific attributes

7.2.2.1 DD_REVISION

Purpose

The DD_REVISION attribute specifies a unique code for the EDD revision of this device, as defined by the manufacturer.

NOTE 1 The manufacturer should change the revision number each time a new device description is issued for a device.

NOTE 2 There can be multiple EDDs for a particular revision of a particular device, in which case the DD_REVISION distinguishes them.

Lexical structure

DD_REVISION integer

The attribute of DD_REVISION is specified in Table 2.

Table 2 – DD_REVISION attribute

Usage	Attribute	Description
m	integer	A two-octet unsigned integer containing the code for a specific EDD file revision.

7.2.2.2 DEVICE_REVISION

Purpose

The DEVICE_REVISION attribute specifies a unique code for the device revision of the field device, as defined by the manufacturer.

Lexical structure

DEVICE_REVISION integer

The attribute of DEVICE_REVISION is specified in Table 3.

Table 3 – DEVICE_REVISION attribute

Usage	Attribute	Description
m	integer	A two-octet unsigned integer containing the code for a specific device revision.

7.2.2.3 DEVICE_TYPE

Purpose

The DEVICE_TYPE attribute specifies an identifier for the device type, as defined by the manufacturer of the device. The identifier should be unique for each type.

NOTE The device type may specify either a category of devices or a specific product.

Lexical structure

DEVICE_TYPE (integer, identifier)

The attribute of DEVICE_TYPE is specified in Table 4.

Table 4 – DEVICE_TYPE attribute

Usage	Attribute	Description
s	integer	A two-octet unsigned integer containing the code for a specific device type or group of device types.
s	identifier	A reference to a two-octet integer which contains the code for a specific device type or group of device types.

NOTE The mapping of the identifier to an integer constant may be stored in an extra file named "devices.cfg". This file contains a list of identifiers for devices or groups of device types.

7.2.2.4 EDD_PROFILE

Purpose

The EDD_PROFILE attribute specifies the selected profile of the EDDL as defined in Annex D.

Lexical structure

EDD_PROFILE integer

The attribute of EDD_PROFILE is specified in Table 5.

Table 5 – EDD_PROFILE attribute

Usage	Attribute	Description
o	integer	A two-octet unsigned integer containing the code for the specific profiles selected from Annex D. The following codes are currently defined: 0x01 Profile for HART Communication Foundation (see D.4) 0x02 Profile for Fieldbus Foundation (see D.3) 0x03 Profile for PROFIBUS (see D.2) 0x04 Profile for PROFINET (see D.2)

7.2.2.5 EDD_VERSION

Purpose

The EDD_VERSION attribute specifies a unique code for the EDD version, which has been used by the manufacturer in constructing the EDD. This attribute shall be included with all device descriptions conforming to this standard.

Lexical structure

EDD_VERSION integer

The attribute of EDD_VERSION is specified in Table 6.

Table 6 – EDD_VERSION attribute

Usage	Attribute	Description
o	integer	A two-octet unsigned integer containing the number 1 for this first edition of the standard.

7.2.2.6 MANUFACTURER

Purpose

The MANUFACTURER attribute identifies the manufacturer. The code allocation should be managed by the specific organizations which are responsible for the different EDDL profiles. The combination of EDD_PROFILE and MANUFACTURER shall be unique.

Lexical structure

MANUFACTURER (long integer, identifier)

The attribute of MANUFACTURER is specified in Table 7.

Table 7 – MANUFACTURER attribute

Usage	Attribute	Description
s	long integer	An unsigned long integer containing the code for a specific manufacturer. If all bits in the value of the integer are set to one, the lexical structure "MANUFACTURER_EXT" shall be present and identifies the manufacturer.

Usage	Attribute	Description
sm	identifier	A reference to an unsigned long integer which contains the code for a specific manufacturer.
NOTE The mapping of the identifier to an integer constant should be stored in an extra file named "devices.cfg". This file contains a list of identifiers for manufacturers.		

7.2.2.7 MANUFACTURER_EXT

Purpose

The MANUFACTURER_EXT attribute is reserved for future use. It is a placeholder for a string providing global identification for the MANUFACTURER.

NOTE The specification of this attribute will be included when it becomes internationally available.

Lexical structure

MANUFACTURER_EXT string

The attribute of MANUFACTURER_EXT is specified in Table 8.

Table 8 – MANUFACTURER_EXT attribute

Usage	Attribute	Description
c	string	A string containing the information for a unique worldwide manufacturer identification.

7.3 AXIS

7.3.1 General structure

Purpose

AXIS describes an axis of a CHART or GRAPH.

Lexical structure

AXIS identifier [CONSTANT_UNIT , HELP, LABEL, MAX_VALUE, MIN_VALUE, SCALING]

The attributes of AXIS are specified in Table 9.

Table 9 – AXIS attributes

Usage	Attribute	Description
o	CONSTANT_UNIT	Lexical element (see 7.3.2.1).
o	HELP	Lexical element (see 7.36.8).
o	LABEL	Lexical element (see 7.36.9).
o	MAX_VALUE	Lexical element (see 7.3.2.1).
o	MIN_VALUE	Lexical element (see 7.3.2.3).
o	SCALING	Lexical element (see 7.3.2.4).

7.3.2 Specific attributes

7.3.2.1 CONSTANT_UNIT

Purpose

The CONSTANT_UNIT attribute is used, if an AXIS has a unit code which never changes. The CONSTANT_UNIT is specified as a string that will be displayed along with the value. An AXIS without a constant unit either has no units associated with it or the units are not constant.

Lexical structure

CONSTANT_UNIT (string)<exp>

The attribute of CONSTANT_UNIT is specified in Table 175.

7.3.2.2 MAX_VALUE

Purpose

The MAX_VALUE attribute specifies the value associated with the last tick mark of the AXIS. The last tick mark of an X_AXIS and Y_AXIS is the right-most and top-most tick mark, respectively. By default, the value associated with the last tick mark of an AXIS without a MAX_VALUE attribute is inferred from the WAVEFORMS on the GRAPH.

Lexical structure

MAX_VALUE (reference, double, float, integer, unsigned_integer)<exp>

The attributes of MAX_VALUE are specified in Table 10.

Table 10 – MAX_VALUE, MIN_VALUE attribute

Usage	Attribute	Description
o	reference	A reference to either a VARIABLE instance, a member of a RECORD instance, an element of a VALUE_ARRAY instance or an element of a REFERENCE_ARRAY instance.
o	double	A double-precision floating-point constant.
o	float	A single-precision floating-point constant.
o	integer	An integer constant.
o	unsigned_integer	An unsigned integer constant.

7.3.2.3 MIN_VALUE

Purpose

The MIN_VALUE attribute specifies the value associated with the first tick mark of the AXIS. The first tick mark of an X_AXIS and Y_AXIS is the left-most and bottom-most tick mark, respectively. By default, the value associated with the first tick mark of an AXIS without a MIN_VALUE attribute is inferred from the WAVEFORMS on the GRAPH.

Lexical structure

MIN_VALUE (reference, double, float, integer, unsigned_integer)<exp>

The attributes of MIN_VALUE are specified in Table 10.

7.3.2.4 SCALING

Purpose

The SCALING attribute specifies the way the tick marks of the AXIS should be scaled. By default, an AXIS without a SCALING attribute is scaled linearly.

Lexical structure

SCALING (LINEAR, LOGARITHMIC)<exp>

The attributes of SCALING are specified in Table 11.

Table 11 – SCALING attribute

Usage	Attribute	Description
s	LINEAR	The tick marks on the axis should be scaled linearly.
s	LOGARITHMIC	The tick marks on the axis should be scaled logarithmically (base 10).

7.4 BLOCK

7.4.1 BLOCK_A

7.4.1.1 General structure

Purpose

A device may be organized in logical blocks. Each BLOCK_A is a logical grouping of PARAMETERS and additional information for BLOCK_A applications. Types of blocks are described by the CHARACTERISTICS attribute. More than one BLOCK_A from the same type can be instantiated.

NOTE This BLOCK_A approach is optimized for access efficiency.

Lexical structure

BLOCK_A identifier [CHARACTERISTICS, LABEL, PARAMETERS, AXIS_ITEMS, CHART_ITEMS, COLLECTION_ITEMS, EDIT_DISPLAY_ITEMS, FILE_ITEMS, GRAPH_ITEMS, GRID_ITEMS, HELP, IMAGE_ITEMS, LIST_ITEMS, MENU_ITEMS, METHOD_ITEMS, PARAMETER_LISTS, REFERENCE_ARRAY_ITEMS, REFRESH_ITEMS, SOURCE_ITEMS, UNIT_ITEMS, WAVEFORM_ITEMS, WRITE_AS_ONE_ITEMS, CHARTS, LISTS, GRAPHS, GRIDS, MENUS, METHODS]

The attributes of BLOCK_A are specified in Table 12.

Table 12 – BLOCK_A attributes

Usage	Attribute	Description
m	CHARACTERISTICS	Lexical element (see 7.4.1.2.1).
m	LABEL	Lexical element (see 7.36.9).
m	PARAMETERS	Lexical element (see 7.4.1.2.2).
o	AXIS_ITEMS	Lexical element (see 7.4.1.2.3).
o	CHART_ITEMS	Lexical element (see 7.4.1.2.4).
o	COLLECTION_ITEMS	Lexical element (see 7.4.1.2.5).
o	EDIT_DISPLAY_ITEMS	Lexical element (see 7.4.1.2.6).
o	FILE_ITEMS	Lexical element (see 7.4.1.2.7).
o	GRAPH_ITEMS	Lexical element (see 7.4.1.2.8).
o	GRID_ITEMS	Lexical elements (see 7.4.1.2.9).
o	HELP	Lexical element (see 7.36.8).
o	IMAGE_ITEMS	Lexical elements (see 7.4.1.2.10).
o	LIST_ITEMS	Lexical element (see 7.4.1.2.11).
o	LOCAL_PARAMETERS	Lexical element (see 7.4.1.2.12).
o	MENU_ITEMS	Lexical element (see 7.4.1.2.13).
o	METHOD_ITEMS	Lexical element (see 7.4.1.2.14).
o	PARAMETER_LISTS	Lexical element (see 7.4.1.2.15).
o	REFERENCE_ARRAY_ITEMS	Lexical element (see 7.4.1.2.16).
o	REFRESH_ITEMS	Lexical element (see 7.4.1.2.17).
o	SOURCE_ITEMS	Lexical element (see 7.4.1.2.18).
o	UNIT_ITEMS	Lexical element (see 7.4.1.2.19).
o	WAVEFORM_ITEMS	Lexical element (see 7.4.1.2.20).
o	WRITE_AS_ONE_ITEMS	Lexical element (see 7.4.1.2.21).
o	CHARTS	Lexical element (see 7.4.1.2.22).
o	LISTS	Lexical element (see 7.4.1.2.23).
o	GRAPHS	Lexical element (see 7.4.1.2.24).
o	GRIDS	Lexical element (see 7.4.1.2.25).

Usage	Attribute	Description
o	MENUS	Lexical element (see 7.4.1.2.26).
o	METHODS	Lexical element (see 7.4.1.2.27).

7.4.1.2 Specific attributes

7.4.1.2.1 CHARACTERISTICS

Purpose

The CHARACTERISTICS attribute specifies the external characteristics for a BLOCK_A. CHARACTERISTICS may include class, function type, block type and execution time. The external characteristics of a BLOCK_A are contained in a RECORD.

NOTE In general the BLOCK_A CHARACTERISTICS fix the type of the BLOCK_A.

Lexical structure

CHARACTERISTICS reference

The attribute of CHARACTERISTICS is specified in Table 13.

Table 13 – CHARACTERISTIC attribute

Usage	Attribute	Description
m	reference	A reference to a RECORD instance.

7.4.1.2.2 PARAMETERS

Purpose

The PARAMETERS attribute contains a list of references to VARIABLE, VALUE_ARRAY or RECORD instances. Each reference shall have an identifier and may have a description and/or a help text.

NOTE The PARAMETERS in a BLOCK_A correspond to individual communication objects, listed in a device object dictionary. The PARAMETERS elements reference the communication definitions of these objects. The object dictionary maps into the specific location for data in a device.

Lexical structure

PARAMETERS [identifier, reference, description, help]+

The attributes of PARAMETERS are specified in Table 14.

Table 14 – PARAMETER attributes

Usage	Attribute	Description
m	identifier	The identifier of the VARIABLE, VALUE_ARRAY or RECORD. Every element of the PARAMETERS list shall have an identifier to be used in the EDD to refer to it.
m	reference	A reference to a VARIABLE, VALUE_ARRAY or RECORD instance.
o	description	A short description for the reference.
o	help	Help text for the reference.

7.4.1.2.3 AXIS_ITEMS

Purpose

The AXIS_ITEMS attribute is a list of the AXIS instances associated with the BLOCK_A.

Lexical structure

AXIS_ITEMS reference+

The attribute of AXIS_ITEMS is specified in Table 15.

Table 15 – AXIS_ITEMS attribute

Usage	Attribute	Description
m	reference	A reference to a AXIS instance.

7.4.1.2.4 CHART_ITEMS**Purpose**

The CHART_ITEMS attribute is a list of the CHART instances associated with the BLOCK_A.

Lexical structure

CHART_ITEMS reference+

The attribute of CHART_ITEMS is specified in Table 16.

Table 16 – CHART_ITEMS attribute

Usage	Attribute	Description
m	reference	A reference to a CHART instance.

7.4.1.2.5 COLLECTION_ITEMS**Purpose**

The COLLECTION_ITEMS attribute is a list of the COLLECTION instances associated with the BLOCK_A.

Lexical structure

COLLECTION_ITEMS reference+

The attribute of COLLECTION_ITEMS is specified in Table 17.

Table 17 – COLLECTION_ITEMS attribute

Usage	Attribute	Description
m	reference	A reference to a COLLECTION instance.

7.4.1.2.6 EDIT_DISPLAY_ITEMS**Purpose**

The EDIT_DISPLAY_ITEMS attribute is a list of the EDIT_DISPLAY instances associated with BLOCK_A.

Lexical structure

EDIT_DISPLAY_ITEMS reference+

The attribute of EDIT_DISPLAY_ITEMS is specified in Table 18.

Table 18 – EDIT_DISPLAY_ITEMS attribute

Usage	Attribute	Description
m	reference	A reference to an EDIT_DISPLAY instance.

7.4.1.2.7 FILE_ITEMS**Purpose**

The FILE_ITEMS attribute is a list of the FILE instances associated with the BLOCK_A.

Lexical structure

FILE_ITEMS reference+

The attribute of FILE_ITEMS is specified in Table 19.

Table 19 – FILE_ITEMS attribute

Usage	Attribute	Description
m	reference	A reference to a FILE instance.

7.4.1.2.8 GRAPH_ITEMS

Purpose

The GRAPH_ITEMS attribute is a list of the GRAPH instances associated with the BLOCK_A.

Lexical structure

GRAPH_ITEMS reference+

The attribute of GRAPH_ITEMS is specified in Table 20.

Table 20 – GRAPH_ITEMS attribute

Usage	Attribute	Description
m	reference	A reference to a GRAPH instance.

7.4.1.2.9 GRID_ITEMS

Purpose

The GRID_ITEMS attribute is a list of the GRID instances associated with the BLOCK_A.

Lexical structure

GRID_ITEMS reference+

The attribute of GRID_ITEMS is specified in Table 21.

Table 21 – GRID_ITEMS attribute

Usage	Attribute	Description
m	reference	A reference to a GRID instance.

7.4.1.2.10 IMAGE_ITEMS

Purpose

The IMAGE_ITEMS attribute is a list of the IMAGE instances associated with BLOCK_A.

Lexical structure

IMAGE_ITEMS reference+

The attribute of IMAGE_ITEMS is specified in Table 22.

Table 22 – IMAGE_ITEMS attribute

Usage	Attribute	Description
m	reference	A reference to a IMAGE instance.

7.4.1.2.11 LIST_ITEMS

Purpose

The LIST_ITEMS attribute is a list of the LIST instances associated with BLOCK_A.

Lexical structure

LIST_ITEMS reference+

The attribute of LIST_ITEMS is specified in Table 23.

Table 23 – LIST_ITEMS attribute

Usage	Attribute	Description
m	reference	A reference to a LIST instance.

7.4.1.2.12 LOCAL_PARAMETERS

Purpose

The LOCAL_PARAMETERS attribute contains a list of references to VARIABLE, VALUE_ARRAY or RECORD instances. Each reference shall have an identifier and may have a description and/or a help text.

NOTE The LOCAL_PARAMETERS in a BLOCK_A do not correspond to communication objects; they are parameters that are only used by the EDD application.

Lexical structure

LOCAL_PARAMETERS [identifier, reference, description, help]+

The attributes of LOCAL_PARAMETERS are specified in Table 14.

7.4.1.2.13 MENU_ITEMS

Purpose

The MENU_ITEMS attribute specifies the MENUs associated with BLOCK_A.

Lexical structure

MENU_ITEMS reference+

The attribute of MENU_ITEMS is specified in Table 24.

Table 24 – MENU_ITEMS attribute

Usage	Attribute	Description
m	reference	A reference to a MENU instance.

7.4.1.2.14 METHOD_ITEMS

Purpose

The METHOD_ITEMS attribute specifies the METHODS associated with BLOCK_A.

Lexical structure

METHOD_ITEMS reference+

The attribute of METHOD_ITEMS is specified in Table 25.

Table 25 – METHOD_ITEMS attribute

Usage	Attribute	Description
m	reference	A reference to a METHOD instance.

7.4.1.2.15 PARAMETER_LISTS

Purpose

The PARAMETER_LISTS attribute specify the VARIABLE_LISTs associated with a BLOCK_A, plus an optional description and help for each VARIABLE_LIST. Every BLOCK_A may contain multiple PARAMETER_LISTS.

Lexical structure

PARAMETER_LISTS [identifier, reference, description, help]+

The attributes of PARAMETER_LISTS are specified in Table 26.

Table 26 – PARAMETER_LISTS attributes

Usage	Attribute	Description
m	identifier	The identifier of the VARIABLE_LIST. Every element of the PARAMETER_LIST list shall have an identifier to be used in the EDD to refer to it.
m	reference	A reference to a VARIABLE_LIST instance.
o	description	A short description of the item.
o	help	Help text for the item.

7.4.1.2.16 REFERENCE_ARRAY_ITEMS

Purpose

The REFERENCE_ARRAY_ITEMS attribute specifies the REFERENCE_ARRAYs associated with the BLOCK_A.

Lexical structure

REFERENCE_ARRAY_ITEMS reference+

The attribute of REFERENCE_ARRAY_ITEMS is specified in Table 27.

Table 27 – REFERENCE_ARRAY_ITEMS attribute

Usage	Attribute	Description
m	reference	A reference to a REFERENCE_ARRAY instance.

7.4.1.2.17 REFRESH_ITEMS

Purpose

The REFRESH_ITEMS attribute specify the REFRESH relations associated with the BLOCK_A.

Lexical structure

REFRESH_ITEMS reference+

The attribute of REFRESH_ITEMS is specified in Table 28.

Table 28 – REFRESH_ITEMS attribute

Usage	Attribute	Description
m	reference	A reference to a REFRESH instance.

7.4.1.2.18 SOURCE_ITEMS

Purpose

The SOURCE_ITEMS attribute specifies the SOURCEs associated with the BLOCK_A.

Lexical structure

SOURCE_ITEMS reference+

The attribute of SOURCE_ITEMS is specified in Table 29.

Table 29 – SOURCE_ITEMS attribute

Usage	Attribute	Description
m	reference	A reference to a SOURCE instance.

7.4.1.2.19 UNIT_ITEMS

Purpose

The UNIT_ITEMS attribute specifies the UNIT relations associated with the BLOCK_A.

Lexical structure

UNIT_ITEMS reference+

The attribute of UNIT_ITEMS is specified in Table 30.

Table 30 – UNIT_ITEMS attribute

Usage	Attribute	Description
m	reference	A reference to an UNIT instance.

7.4.1.2.20 WAVEFORM_ITEMS

Purpose

The WAVEFORM_ITEMS attribute specifies the WAVEFORMs associated with the BLOCK_A.

Lexical structure

WAVEFORM_ITEMS reference+

The attribute of WAVEFORM_ITEMS is specified in Table 31.

Table 31 – WAVEFORM_ITEMS attribute

Usage	Attribute	Description
m	reference	A reference to a WAVEFORM instance.

7.4.1.2.21 WRITE_AS_ONE_ITEMS

Purpose

The WRITE_AS_ONE_ITEMS attribute specifies the WRITE_AS_ONE relations associated with the BLOCK_A.

Lexical structure

WRITE_AS_ONE_ITEMS reference+

The attribute of WRITE_AS_ONE_ITEMS is specified in Table 32.

Table 32 – WRITE_AS_ONE_ITEMS attribute

Usage	Attribute	Description
m	reference	A reference to a WRITE_AS_ONE instance.

7.4.1.2.22 CHARTS

Purpose

The CHARTS attribute specifies the CHARTs associated with the BLOCK_A. Only those CHARTs that will be referenced via a specific BLOCK_A instance (see 7.38.21) need be specified.

Lexical structure

CHARTS (identifier, reference, description, help)+

The attribute of CHARTS is specified in Table 33.

Table 33 – CHARTS attribute

Usage	Attribute	Description
m	identifier	The name by which the CHART may be referenced.
m	reference	A reference to a CHART instance.
o	description	A short description of the CHART.
o	help	Help text for the CHART.

7.4.1.2.23 LISTS**Purpose**

The LISTS attribute specifies the LISTS associated with the BLOCK_A. Only those LISTS that will be referenced via a specific BLOCK_A instance (see 7.4.1) need be specified.

Lexical structure

LISTS (identifier, reference, description, help)+

The attribute of LISTS is specified in Table 34.

Table 34 – LISTS attribute

Usage	Attribute	Description
m	identifier	The name by which the LIST may be referenced.
m	reference	A reference to a LIST instance.
o	description	A short description of the LIST.
o	help	help text for the LIST.

7.4.1.2.24 GRAPHS**Purpose**

The GRAPHS attribute specifies the GRAPHS associated with the BLOCK_A. Only those GRAPHS that will be referenced via a specific BLOCK_A instance (see 7.38.23) need be specified.

Lexical structure

GRAPHS (identifier, reference, description, help)+

The attribute of GRAPHS is specified in Table 35.

Table 35 – GRAPHS attribute

Usage	Attribute	Description
m	identifier	The name by which the GRAPH may be referenced.
m	reference	A reference to a GRAPH instance.
o	description	A short description of the GRAPH.
o	help	Help text for the GRAPH.

7.4.1.2.25 GRIDS**Purpose**

The GRIDS attribute specifies the GRIDS associated with the BLOCK_A. Only those GRIDS that will be referenced via a specific BLOCK_A instance (see 7.38.24) need be specified.

Lexical structure

GRIDS (identifier, reference, description, help)+

The attribute of GRIDS is specified in Table 36.

Table 36 – GRIDS attribute

Usage	Attribute	Description
m	identifier	The name by which the GRID may be referenced.
m	reference	A reference to a GRID instance.
o	description	A short description of the GRID.
o	help	Help text for the GRID.

7.4.1.2.26 MENUS**Purpose**

The MENUS attribute specifies the MENUS associated with the BLOCK_A. Only those MENUS that will be referenced via a specific BLOCK_A instance (see 7.38.25) need be specified.

Lexical structure

MENUS (identifier, reference, description, help)+

The attribute of MENUS is specified in Table 37.

Table 37 – MENUS attribute

Usage	Attribute	Description
m	identifier	The name by which the MENU may be referenced.
m	reference	A reference to a MENU instance.
o	description	A short description of the MENU.
o	help	Help text for the MENU.

7.4.1.2.27 METHODS**Purpose**

The METHODS attribute specifies the METHODS associated with the BLOCK_A. Only those METHODS that will be referenced via a specific BLOCK_A instance (see 7.38.26) need be specified.

Lexical structure

METHODS (identifier, reference, description, help)+

The attribute of METHODS is specified in Table 38.

Table 38 – METHODS attribute

Usage	Attribute	Description
m	identifier	The name by which the METHOD may be referenced.
m	reference	A reference to a METHOD instance.
o	description	A short description of the METHOD.
o	help	Help text for the METHOD.

7.4.2 BLOCK_B**7.4.2.1 General structure****Purpose**

The variables of a device or module respectively are structured in blocks corresponding to device components or functional parts. Three types of blocks are defined: PHYSICAL, TRANSDUCER and FUNCTION. More than one BLOCK_B from the same TYPE can be

instantiated. For efficient access, each instance has its own NUMBER. For each of the block types there are different threads of NUMBER sequences, starting with 1.

When accessing a VARIABLE within a device, the BLOCK_B construct is used in conjunction with the COMMAND to provide relative addressing.

NOTE 1 This BLOCK_B approach is optimized for memory efficiency.

NOTE 2 The storage of a variable in a device is manufacturer specific and is represented by a device directory. A device directory contains a summary of the available BLOCK_B entries in a device. Furthermore, the device directory contains for each BLOCK_B a pointer to its physical address. An EDD application finds a single object by adding an offset to the BLOCK_B physical address.

NOTE 3 Within a BLOCK_B instance, relative indexing is used. This relative index is defined in the device profile.

Lexical structure

BLOCK_B identifier [NUMBER, TYPE]

The attributes of BLOCK_B are specified in Table 39.

Table 39 – BLOCK_B attributes

Usage	Attribute	Description
m	NUMBER	Lexical element (see 7.4.2.2.1).
m	TYPE	Lexical element (see 7.4.2.2.2).

7.4.2.2 Specific attributes

7.4.2.2.1 NUMBER

Purpose

The NUMBER attribute enumerates each BLOCK_B of the same TYPE in the device. A device may contain several BLOCK_B instances, which are identified with the NUMBER attribute.

Lexical structure

NUMBER (integer, expression) <exp>

The attribute of NUMBER is specified in Table 40.

Table 40 – NUMBER attributes

Usage	Attribute	Description
s	integer	The number of the BLOCK_B instance of the same TYPE.
s	expression	An expression which shall be evaluated to a non-negative integer that is within the index range of the NUMBER attribute (for the description of expressions, see 7.39.7).

7.4.2.2.2 TYPE

Purpose

The TYPE attribute is used to specify one of the three block types.

Lexical structure

TYPE [PHYSICAL, TRANSDUCER, FUNCTION]

The attributes of TYPE are specified in Table 41.

Table 41 – TYPE attributes

Usage	Attribute	Description
s	FUNCTION	A named block consisting of one or more input, output and contained parameters. Function blocks represent the basic automation functions performed by an EDD application, which is independent of the specific devices and the network. Each function block processes input parameters according to a specified algorithm and an internal set of contained parameters. They produce output parameters that are available for use within the same function block EDD application or by other function block applications.
s	PHYSICAL	Hardware/resource specific characteristics of a field device are made visible through the physical block. Similar to transducer blocks, they isolate function blocks from the physical hardware by containing a set of implementation independent hardware/resource parameters.
s	TRANSDUCER	Transducer blocks isolate function blocks from the specifics of I/O devices, such as sensors, actuators, and switches. Transducer blocks control access to I/O devices through a device-independent interface defined for use by function blocks. Transducer blocks perform functions, such as calibration and linearization, on I/O data to convert it to a device independent representation.

7.5 CHART

7.5.1 General structure

Purpose

CHART describes a chart used to display data from a device in the EDD application. A CHART is used to display continuous data values from the device, as opposed to a GRAPH that is used to display a data set that is stored in a device. The data from the device may be adjusted prior to being displayed via the use of INIT_ACTIONS and RERESH_ACTIONS on the SOURCE members of the CHART.

Lexical structure

CHART identifier [MEMBERS, CYCLE_TIME, HEIGHT, WIDTH, LENGTH, HELP, LABEL, TYPE, VALIDITY]

The attributes of CHART are specified in Table 42.

Table 42 – CHART attributes

Usage	Attribute	Description
M	MEMBERS	Lexical element (see 7.36.12).
o	CYCLE_TIME	Lexical element (see 7.5.2.1).
o	HEIGHT	Lexical element (see 7.36.7).
o	HELP	Lexical element (see 7.36.8).
o	LABEL	Lexical element (see 7.36.9).
o	LENGTH	Lexical element (see 7.5.2.2).
o	TYPE	Lexical element (see 7.5.2.3).
o	VALIDITY	Lexical element (see 7.36.16).
o	WIDTH	Lexical element (see 7.36.15).

7.5.2 Specific attributes

7.5.2.1 CYCLE_TIME

Purpose

The CYCLE_TIME attribute specifies the length of time, in ms, the EDD application shall wait between successive reads of the source values in the device. By default, the cycle time of a CHART without a CYCLE_TIME attribute is 1 000 ms.

Lexical structure

CYCLE_TIME (integer)<exp>

The attribute of CYCLE_TIME is specified in Table 43.

Table 43 – CYCLE_TIME attribute

Usage	Attribute	Description
m	integer	The length of time, in ms, between successive reads of the source values in the device.

7.5.2.2 LENGTH

Purpose

The LENGTH attribute specifies the length of time, in ms, that is displayed by the CHART. The number of samples displayed by a chart can be calculated by dividing the LENGTH attribute by the CYCLE_TIME attribute. By default, the cycle time of a CHART without a LENGTH attribute is 600 000 ms.

Lexical structure

LENGTH (integer)<exp>

The attribute of LENGTH is specified in Table 44.

Table 44 – LENGTH attribute

Usage	Attribute	Description
m	integer	The length of time, in ms, that is displayed by the CHART.

7.5.2.3 TYPE

Purpose

The TYPE attribute specifies the type of CHART the EDD shall display. By default, the type of a CHART without a TYPE attribute is STRIP.

Lexical structure

TYPE (GAUGE, HORIZONTAL_BAR, SCOPE, STRIP, SWEEP, VERTICAL_BAR)<exp>

The attributes of TYPE are specified in Table 45.

Table 45 – TYPE attribute

Usage	Attribute	Description
s	GAUGE	A single source value is displayed as a gauge, which is a graphical representation of the data similar to the fuel gauge in an automobile.
s	HORIZONTAL_BAR	The source values are displayed as a horizontal bar chart.
s	SCOPE	When the source values reach the end of the display area of the CHART the display area is erased and the new source values are once again displayed starting at the beginning of the display area.
s	STRIP	When the source values reach the end of the display area of the CHART, the display area is scrolled with the oldest source values being removed from the display area and the newest source values being added to the display area.
s	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, but unlike SCOPE only the portion of the display area needed to display the new source values is erased.
s	VERTICAL_BAR	The source values are displayed as a vertical bar chart.

7.6 COLLECTION

7.6.1 General structure

Purpose

A COLLECTION is a logical group of items, such as VARIABLES or MENUS. An identifier is assigned to each item. The items may be referenced within the device description by using the COLLECTION identifier and the item name.

NOTE 1 COLLECTION is intended to be used by a tool, for example, to identify parameters that should be processed together.

NOTE 2 The recommended supported nesting level is two (for example, COLLECTION of COLLECTION).

Lexical structure

COLLECTION identifier, item-type, [MEMBERS, HELP, LABEL, VALIDITY]

The attributes of COLLECTION are specified in Table 46.

Table 46 – COLLECTION attributes

Usage	Attribute	Description
o	item-type	Lexical element (see 7.6.2).
m	MEMBERS	Lexical element (see 7.36.10).
o	HELP	Lexical element (see 7.36.8).
o	LABEL	Lexical element (see 7.36.9).
o	VALIDITY	Lexical element (see 7.36.16).

7.6.2 Specific attribute – item-type

Purpose

The item-type attribute specifies the type of members in the collection. If item-type is specified, all the collection members shall be of the specified type. Otherwise, the collection members may be of any type.

Lexical structure

item-type

Table 47 shows the allowed item-types:

Table 47 – item-type

Usage	item-type	Description
s	AXIS	Selection of this basic construct.
s	BLOCK_A	Selection of this basic construct.
s	BLOCK_B	Selection of this basic construct.
s	CHART	Selection of this basic construct.
s	COLLECTION	Selection of this basic construct.
s	COMMAND	Selection of this basic construct.
s	COMPONENT	Selection of this basic construct.
s	COMPONENT_FOLDER	Selection of this basic construct.
s	COMPONENT_REFERENCE	Selection of this basic construct.
s	COMPONENT_RELATION	Selection of this basic construct.
s	CONNECTION	Selection of this basic construct.
s	DOMAIN	Selection of this basic construct.
s	EDIT_DISPLAY	Selection of this basic construct.
s	FILE	Selection of this basic construct.
s	GRAPH	Selection of this basic construct.
s	GRID	Selection of this basic construct.

Usage	item-type	Description
s	IMAGE	Selection of this basic construct.
s	INTERFACE	Selection of this basic construct.
s	LIST	Selection of this basic construct.
s	MENU	Selection of this basic construct.
s	METHOD	Selection of this basic construct.
s	PROGRAM	Selection of this basic construct.
s	RECORD	Selection of this basic construct.
s	REFERENCE_ARRAY	Selection of this basic construct.
s	REFRESH	Selection of this basic construct.
s	RESPONSE_CODES	Selection of this basic construct.
s	SOURCE	Selection of this basic construct.
s	TEMPLATE	Selection of this basic construct.
s	UNIT	Selection of this basic construct.
s	VALUE_ARRAY	Selection of this basic construct.
s	VARIABLE	Selection of this basic construct.
s	VARIABLE_LIST	Selection of this basic construct.
s	WAVEFORM	Selection of this basic construct.
s	WRITE_AS_ONE	Selection of this basic construct.

7.7 COMMAND

7.7.1 General structure

Purpose

COMMAND is a construct to support mapping of the EDD communication elements to a chosen communication system. It specifies various elements required to build a communication frame. If this construct is used by an EDDL profile, every data item needing to be transmitted shall be referenced inside a COMMAND and also in the upload-/download MENU (see 7.23.2.9).

NOTE 1 The address field of a communication frame is specified using the selectable constructs BLOCK, SLOT, SUB_SLOT, NUMBER, INDEX, MODULE as applicable. If multiple EDD application process instances, or one EDD application process instance which needs an explicit reference, exist in a device, these are specified by the additional construct CONNECTION. The type of the communication frame or its control field is specified by the OPERATION construct. The data content of the communication frame is specified by the TRANSACTION construct.

NOTE 2 The scope of BLOCK, SLOT, SUB_SLOT, NUMBER is limited to a given EDD application process instance, i.e. the same BLOCK/SLOT/NUMBER value may be used in conjunction with different CONNECTION values to identify different data items within different EDD application process instances.

Lexical structure

COMMAND identifier, [OPERATION, TRANSACTION+, INDEX, BLOCK_B, NUMBER, SLOT, SUB_SLOT, CONNECTION, HEADER, MODULE, RESPONSE_CODES]

The attributes of COMMAND are specified in Table 48.

Table 48 – COMMAND attributes

Usage	Attribute	Description
m	OPERATION	Lexical element (see 7.7.2.1).
m	TRANSACTION	Lexical element (see 7.7.2.2).
c	INDEX	Lexical element (see 7.7.2.3).
s	BLOCK_B	Lexical element (see 7.7.2.4).
s	NUMBER	Lexical element (see 7.7.2.5).
s	SLOT	Lexical element (see 7.7.2.6).
s	SUB_SLOT	Lexical element (see 7.7.2.7).
o	CONNECTION	Lexical element (see 7.7.2.8).

Usage	Attribute	Description
o	HEADER	Lexical element (see 7.7.2.9).
o	MODULE	Lexical element (see 7.7.2.10).
o	RESPONSE_CODES	Lexical element (see 7.36.13).

NOTE 1 Response codes appearing within a transaction apply only to that transaction.

NOTE 2 Response codes appearing outside of any transaction apply to all transactions.

NOTE 3 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.

7.7.2 Specific attributes

7.7.2.1 OPERATION

Purpose

The OPERATION attribute specifies the actions taken by the field device upon receiving a service request from the communication system.

Lexical structure

OPERATION [(COMMAND, DATA_EXCHANGE, READ, string, WRITE)]

The attributes of OPERATION are specified in Table 49.

Table 49 – OPERATION attribute

Usage	Attribute	Description
s	COMMAND	The device performs a predefined set of actions (for example, self-test, master reset). The actions to be performed are device-specific and are identified by the NUMBER attribute.
s	DATA_EXCHANGE	Specifies a bi-directional transaction. The input data is specified in the REPLY attribute and the output data is specified in the REQUEST attribute of the TRANSACTION.
s	READ	Specifies a read transaction. The field device returns the current values of the specified variables. VARIABLES shall be defined in the REPLY section of TRANSACTION.
s	string	Specifies manufacturer specific services for transactions. The transaction to be performed should be explained in a corresponding specification.
s	WRITE	Specifies a write transaction. The field device receives the current values of the specified variables. VARIABLES shall be defined in the REQUEST section of TRANSACTION.

7.7.2.2 TRANSACTION

7.7.2.2.1 General structure

Purpose

Transactions specify the data field of the command's request and reply messages. It is possible to define more than one transaction. The syntax should support a unique identification of the TRANSACTION. Each TRANSACTION may include a set of response codes that apply only to that particular transaction.

EXAMPLE Block commands, such as commands 4 and 5 of HART⁵ Universal Command Revision 4 or later, are examples of multiple transaction commands.

Lexical structure

TRANSACTION [REPLY, REQUEST, integer, RESPONSE_CODES]

⁵ See Annex D.4.

The attributes of TRANSACTION are specified in Table 50.

Table 50 – TRANSACTION attributes

Usage	Attribute	Description
m	REPLY	Lexical element (see 7.7.2.2.2.1).
m	REQUEST	Lexical element (see 7.7.2.2.2.2).
o	integer	The number of the TRANSACTIONS (if more than one TRANSACTION is used).
o	RESPONSE_CODES	Lexical element (see 7.36.13).

7.7.2.2.2 Specific attributes

7.7.2.2.2.1 REPLY

Purpose

The REPLY attribute contains a list of data items (constants or references to variables, lists, arrays, or collections), which are received from the device. The order of the list shall be maintained in the corresponding data field in the communicated PDU. If the referenced data items are not of type VARIABLE, the referenced data inside the EDD elements shall always terminate as references of VARIABLE. All other referenced EDD element types are not allowed. The combination of list, array, and collection with item mask is not allowed.

When a VARIABLE does not begin and end on octet boundaries, a mask should be used to specify how the variable is packed into the data field.

Every bit in the mask may have a semantical meaning. In addition, if the least significant bit is set in an item mask, the next data item (if any) is contained in the following octet(s) of the PDU. If the least significant bit is clear (not set), the next data item is contained in the same octet(s) of the PDU.

When a LIST is referenced, the EDD application shall reset the LIST and after that it shall fill the LIST with the data of the received PDU.

NOTE 1 Even if the least significant bit is not used for actual data, it should be set in an item mask (associated with a "dummy" data item) if data items follow.

NOTE 2 Examples are given in Annex C.

Lexical structure

REPLY [reference, item-mask, INFO, INDEX]+

The attributes of REPLY are specified in Table 51.

Table 51 – REPLY and REQUEST attributes

Usage	Attribute	Description
m	reference	A reference to a constant or a VARIABLE, ARRAY, COLLECTION, REFERENCE_ARRAY, or LIST instance.
o	item-mask	A bit pattern to extract the VARIABLE from the data field. The item-mask shall be used only for the data types INTEGER, UNSIGNED_INTEGER, ENUMERATED and BIT_ENUMERATED. If the least significant bit is set in the item-mask, the following item in the REQUEST or REPLY list is to be used. The length of the item-mask shall be 1 byte to 4 bytes and shall correspond to the length of the data item in the REQUEST or REPLY list.
o	INFO	The VARIABLE is not actually stored in the device. It provides additional information to ensure correct processing.
o	INDEX	The VARIABLE is used in the REQUEST or REPLY as an index into an REFERENCE_ARRAY.

NOTE 3 A variable may be qualified with both, INDEX and INFO in which case it is called a local index variable. Commands with local indices behave exactly the same as commands with INDEX qualified variables except the index variables are not stored in the device.

NOTE 4 INFO may be used to send or read a VARIABLE with units differing from those used in the device.

7.7.2.2.2 REQUEST

Purpose

The REQUEST attribute contains a list of data items (constants or references to variables, lists, arrays, or collections), which are sent to the device. The order of the list shall be maintained in the corresponding data field in the communicated PDU. If the referenced data items are not of type VARIABLE, the referenced data inside the EDD elements shall always terminate as references of VARIABLE. All other referenced EDD element types are not allowed. The combination of list, array, and collection with item mask is not allowed.

When a data item does not begin and end on octet boundaries of the PDU, an item mask shall be used to specify how the data item is packed into the data field. The item mask is an integer number of octets that shall be logically ANDed with the communicated PDU to extract the desired data item.

If no data item mask is specified, an implicit mask is used with all bits set for every octet, corresponding to the data item length.

Every bit in the mask may have a semantic meaning. In addition, if the least significant bit is set in an item mask, the next data item (if any) is contained in the following octet(s) of the PDU. If the least significant bit is clear (not set), the next data item is contained in the same octet(s) of the PDU.

NOTE 1 Even if the least significant bit is not used for actual data it should be set in an item mask (associated with a “dummy” data item) if data items follow.

NOTE 2 Examples are given in Annex C.

Lexical structure

REQUEST integer, [reference, item-mask, INFO, INDEX]+

The attributes of REQUEST are specified in Table 51.

7.7.2.3 INDEX

Purpose

The INDEX attribute shall be present if the SLOT or BLOCK_B attribute is used. The INDEX specifies the data item address within the SLOT or the BLOCK_B.

Lexical structure

INDEX (integer, reference, expression)<exp>

The attributes of INDEX are specified in Table 52.

Table 52 – INDEX attribute

Usage	Attribute	Description
s	integer	The number of the index.
s	reference	A reference to a VARIABLE instance containing the index.
s	expression	An expression which shall be evaluated to a non-negative integer that is within the index range of the INDEX attribute (for the description of expressions, see 7.40).

7.7.2.4 BLOCK_B

Purpose

The BLOCK_B attribute provides a reference to an individual block within the device.

Lexical structure

BLOCK_B (reference)<exp>

The attribute of BLOCK_B is specified in Table 53.

Table 53 – BLOCK_B attribute

Usage	Attribute	Description
m	reference	A reference to a BLOCK_B instance.

7.7.2.5 NUMBER

Purpose

The NUMBER attribute is used to enumerate the COMMANDS.

Lexical structure

NUMBER (integer, reference, expression)<exp>

The attribute of NUMBER is specified in Table 54.

Table 54 – NUMBER attribute

Usage	Attribute	Description
s	integer	The number of the COMMAND.
s	reference	A reference to a VARIABLE instance containing the number of the COMMAND.
s	expression	An expression which shall be evaluated to a non-negative integer that is within the index range of the NUMBER attribute (for the description of expressions, see 7.40).

7.7.2.6 SLOT

Purpose

The SLOT attribute specifies the number of a slot. Data items of a device may be allocated to logical slots. Within a slot each data item is addressed using an index.

Lexical structure

SLOT (integer, reference, expression)<exp>

The attributes of SLOT are specified in Table 55.

Table 55 – SLOT attribute

Usage	Attribute	Description
s	integer	The number of the slot.
s	reference	A reference to a VARIABLE instance containing the slot number.
s	expression	An expression, which shall be evaluated to a non-negative integer that is within the index range of the SLOT attribute (for the description of expressions, see 7.40).

7.7.2.7 SUB_SLOT

Purpose

The SUB_SLOT attribute specifies the number of a sub-slot. Data items of a device may be allocated to logical sub-slots. Within a sub-slot each data item is addressed using an index.

Lexical structure

SUB_SLOT (integer, reference, expression)<exp>

The attributes of SUB_SLOT are specified in Table 56.

Table 56 – SUB_SLOT attribute

Usage	Attribute	Description
s	integer	The number of the sub-slot.
s	reference	A reference to a VARIABLE instance containing the sub-slot number.
s	expression	An expression, which shall be evaluated to a non-negative integer that is within the index range of the SUB_SLOT attribute (for the description of expressions, see 7.40).

7.7.2.8 CONNECTION**Purpose**

The CONNECTION attribute provides a reference to the basic construct CONNECTION which contains the number of an EDD application process instance.

Lexical structure

CONNECTION (reference) <exp>

The attribute of CONNECTION is specified in Table 57.

Table 57 – CONNECTION attribute

Usage	Attribute	Description
m	reference	A reference to a CONNECTION instance specifying the EDD application.

7.7.2.9 HEADER**Purpose**

The HEADER attribute is provided for specific communication protocols.

NOTE It may be used for addressing data in a device or for a reset operation.

Lexical structure

HEADER (string) <exp>

The attribute of HEADER is specified in Table 58.

Table 58 – HEADER attribute

Usage	Attribute	Description
m	string	A string that is used for the communication with specific communication protocols.

7.7.2.10 MODULE**Purpose**

The MODULE attribute provides an external reference. Use of this attribute is out of the scope of this standard.

NOTE The MODULE is included for compatibility reasons with existing devices that use other description files.

Lexical structure

MODULE (reference) <exp>

The attribute of MODULE is specified in Table 59.

Table 59 – MODULE attribute

Usage	Attribute	Description
m	reference	A reference to a MODULE instance in an external description file.

7.8 COMPONENT

7.8.1 General Structure

Purpose

Describes the configuration behavior of the device that is described in the EDD. A component enumerates all relations to other components.

A component consists of one DDL file, import files, include files, dictionaries, images, and help files.

All modules of a modular device shall be described with COMPONENTs. The relations between the different components are described with COMPONENT_RELATIONS. The hierarchy of the catalog is described with COMPONENT_FOLDERS and COMPONENTs. COMPONENT.

Conditional expressions can have access to different components using OBJECT_REFERENCES. OBJECT_REFERENCES allows the access to any variables and allows to call methods.

Lexical structure

COMPONENT identifier [LABEL, CAN_DELETE, CHECK_CONFIGURATION, CLASSIFICATION, COMPONENT_PARENT, COMPONENT_PATH, COMPONENT_RELATIONS, DECLARATION, DETECT, EDD, HELP, INITIAL_VALUES, PROTOCOL, REDUNDANCY, SCAN, SCAN_LIST, SUPPLIED_INTERFACE]

The attributes are specified in Table 60.

Table 60 – COMPONENT attribute

Usage	Attribute	Description
m	LABEL	Lexical element (see 7.36.9).
o	CAN_DELETE	Lexical element (see 7.8.2.1).
o	CHECK_CONFIGURATION	Lexical element (see 7.8.2.2).
o	CLASSIFICATION	Lexical element (see 7.36.1).
o	COMPONENT_PARENT	Lexical element (see 7.36.2).
o	COMPONENT_PATH	Lexical element (see 7.36.3).
o	COMPONENT_RELATIONS	Lexical element (see 7.8.2.3).
o	DECLARATION	Lexical element (see 7.8.2.4).
o	DETECT	Lexical element (see 7.8.2.5).
o	EDD	Lexical element (see 7.8.2.6).
o	HELP	Lexical element (see 7.36.8).
o	INITIAL_VALUES	Lexical element (see 7.8.2.7).
o	PROTOCOL	Lexical element (see 7.36.13).
o	REDUNDANCY	Lexical element (see 7.8.2.8).
o	SCAN	Lexical element (see 7.8.2.9).
o	SCAN_LIST	Lexical element (see 7.8.2.10).
o	SUPPLIED_INTERFACE	Lexical element (see 7.36.15).

7.8.2 Specific attributes

7.8.2.1 CAN_DELETE

Purpose

The CAN_DELETE attribute resolves to a TRUE or FALSE that indicates whether it is safe to delete this module instance from a modular device. This attribute is used in online and offline

use cases. By default, CAN_DELETE is TRUE. An example thereof is: channels of a module can not be deleted; only the module in whole can be deleted.

Lexical structure

CAN_DELETE (FALSE, TRUE) <exp>

The attributes of CAN_DELETE are specified in Table 61.

Table 61 – CAN_DELETE attribute

Usage	Attribute	Description
s	TRUE	The component can be deleted.
s	FALSE	The component can not be deleted.

7.8.2.2 CHECK_CONFIGURATION

Purpose

The CHECK_CONFIGURATION attribute reference a method that shall be called by the EDD Application to confirm the current configuration of this COMPONENT. A BI_SUCCESS return value indicates that the configuration is valid; all other values indicate that the configuration is invalid. If the method aborts, then the configuration is invalid. The referenced method shall have no input parameters and single output parameter of type DD_STRING that returns a message. This message shall be displayed to the user by the EDD Application. This message may contain several reasons for an invalid configuration. In this case, the message can contain carriage return and line feed. This method may be used for both online and offline.

Lexical structure

CHECK_CONFIGURATION (NULL, string) <exp>

The attributes of CHECK_CONFIGURATION are specified in Table 62.

Table 62 – CHECK_CONFIGURATION attribute

Usage	Attribute	Description
s	NULL	The configuration is correct.
s	string	A string typically generated by a METHOD. That string includes the problem description and may be a hint to solve the problem.

7.8.2.3 COMPONENT_RELATIONS

Purpose

The COMPONENT_RELATIONS attribute lists the relations to other components that this modular device or module can support. This EDD can list sibling-modules child-modules and parent-modules.

Lexical structure

COMPONENT_RELATIONS [(component-relation-reference) <exp>]+

The attributes of COMPONENT_RELATIONS are specified in Table 63.

Table 63 – COMPONENT_RELATIONS attribute

Usage	Attribute	Description
m	component-relation-reference	A list of component relation identifiers. Conditional expressions are allowed.

7.8.2.4 DECLARATION

Purpose

The DECLARATION attribute specifies basic constructs associated with the component.

Lexical structure

DECLARATION [item]+

The attributes of DECLARATION are specified in Table 64.

Table 64 – DECLARATION attribute

Usage	Attribute	Description
m	item	Declarations of any basic constructs (see 7.1.4).

7.8.2.5 DETECT

Purpose

The DETECT attribute indicates the method that should be called by the EDD application to detect a component.

NOTE In HART the EDD application should take care to read with command 75 to identify a component.

Lexical structure

DETECT method_reference

The attribute of DETECT is specified in Table 65.

Table 65 – DETECT attribute

Usage	Attribute	Description
s	method_reference	A reference to a detect method.

7.8.2.6 EDD

Purpose

The EDD attribute specifies a file that contains the rest of the EDD. This is only required if the component description is in a separate file.

Lexical structure

EDD string

The attributes of EDD are specified in Table 66.

Table 66 – EDD attribute

Usage	Attribute	Description
m	string	The file that contains the rest of the EDD.

7.8.2.7 INITIAL_VALUES

Purpose

The INITIAL_VALUES attribute is a list of initialization for variables. Even if the variables do have an INITIAL_VALUE or DEFAULT_VALUE, these INITIAL_VALUES are taken for initialization.

It could be used for example to define different initialization for different usages like differential pressure, level and flow measurement for a pressure transmitter.

Lexical structure

INITIAL_VALUES [reference, value<exp>]+

The attributes of CHECK_CONFIGURATION are specified in Table 62.

Table 67 – CHECK_CONFIGURATION attribute

Usage	Attribute	Description
s	reference	A reference to a variable.
s	value	An initial value of the referenced variable.

7.8.2.8 REDUNDANCY**Purpose**

The REDUNDANCY attribute resolves to an integer (defaults to one) that indicates how many redundant sockets are supported by this device.

Lexical structure

REDUNDANCY (integer)<exp>

The attributes of REDUNDANCY are specified in Table 68.

Table 68 – REDUNDANCY attribute

Usage	Attribute	Description
o	integer	The number of Fieldbus sockets.

7.8.2.9 SCAN**Purpose**

The SCAN attribute indicates the method that should be called by the EDD application to create a life list of a modular device.

NOTE In HART the EDD application should take care to read with command 74 the list of the existing modules.

Lexical structure

SCAN method_reference

The attribute of SCAN is specified in Table 69.

Table 69 – SCAN attribute

Usage	Attribute	Description
s	method_reference	A reference to the scan method.

7.8.2.10 SCAN_LIST**Purpose**

The SCAN_LIST attribute is a reference to the result list of sub-components that is created by the scan method.

NOTE In HART the EDD application should take care to read with command 74 the list of the existing modules.

Lexical structure

SCAN_LIST reference

The attribute of SCAN_LIST is specified in Table 70.

Table 70 – SCAN_LIST attribute

Usage	Attribute	Description
s	reference	A reference to the scan list.

7.9 COMPONENT_FOLDER

Purpose

A COMPONENT_FOLDER describes a folder in a catalog. Folders can include other folders. Folders allow to group components e.g. components of a product family. The LABELS of the folders are associated with the COMPONENT_PATH.

Lexical structure

COMPONENT_FOLDER identifier [LABEL, CLASSIFICATION, COMPONENT_PARENT, COMPONENT_PATH, HELP, PROTOCOL]

The attributes of COMPONENT_FOLDER are specified in Table 71.

Table 71 – COMPONENT_FOLDER attribute

Usage	Attribute	Description
m	LABEL	Lexical element (see 7.36.9).
o	CLASSIFICATION	Lexical element (see 7.36.1).
o	COMPONENT_PARENT	Lexical element (see 7.36.2).
o	COMPONENT_PATH	Lexical element (see 7.36.3).
o	HELP	Lexical element (see 7.36.8).
o	PROTOCOL	Lexical element (see 7.36.15).

7.10 COMPONENT_REFERENCE

Purpose

A COMPONENT_REFERENCE refers to another component via the COMPONENT_PARENT attribute or with of the PROTOCOL, CLASSIFICATION, MANUFACTURER and COMPONENT_PATH attributes. If a folder is referenced, all components that are included in this folder are referenced.

If the COMPONENT_REFERENCE resolves to a folder containing a single EDD, then the COMPONENT_REFERENCE is for a single device. Otherwise a family of devices is indicated.

Alternative to CLASSIFICATION it is possible to reference a component with DEVICE_TYPE and DEVICE_REVISION.

Lexical structure

COMPONENT_REFERENCE identifier [CLASSIFICATION, COMPONENT_PARENT, COMPONENT_PATH, DEVICE_REVISION, DEVICE_TYPE, MANUFACTURER, PROTOCOL]

The attributes of COMPONENT_FOLDER are specified in Table 72.

Table 72 – COMPONENT_REFERENCE attribute

Usage	Attribute	Description
o	CLASSIFICATION	Lexical element (see 7.36.1).
o	COMPONENT_PARENT	Lexical element (see 7.36.2).
o	COMPONENT_PATH	Lexical element (see 7.36.3).
o	DEVICE_REVISION	Lexical element (see 7.2.2.2).
o	DEVICE_TYPE	Lexical element (see 7.2.2.3).
o	MANUFACTURER	Lexical element (see 7.2.2.6), defaults MANUFACTURER of EDD.
o	PROTOCOL	Lexical element (see 7.36.13).

7.11 COMPONENT_RELATION

7.11.1 General Structure

Purpose

A COMPONENT_RELATION specifies a relation between a module and one or more other modules and the constraints that shall be met to instantiate those other modules.

Lexical structure

COMPONENT_RELATION identifier [COMPONENTS, LABEL, RELATION_TYPE, ADDRESSING, HELP, MAXIMUM_NUMBER, MINIMUM_NUMBER, REQUIRED_INTERFACE, SUPPLIED_INTERFACE]

The attributes of COMPONENT_RELATION are specified in Table 73.

Table 73 – COMPONENT_RELATION attribute

Usage	Attribute	Description
m	COMPONENTS	Lexical element (see 7.11.2.1).
m	LABEL	Lexical element (see 7.36.9).
m	RELATION_TYPE	Lexical element (see 7.11.2.2).
o	ADDRESSING	Lexical element (see 7.11.2.3).
o	HELP	Lexical element (see 7.36.8).
o	MAXIMUM_NUMBER	Lexical element (see 7.11.2.4).
o	MINIMUM_NUMBER	Lexical element (see 7.11.2.5).
o	REQUIRED_INTERFACE	Lexical element (see 7.11.2.6).
o	SUPPLIED_INTERFACE	Lexical element (see 7.36.15).

7.11.2 Specific attributes

7.11.2.1 COMPONENTS

Purpose

The COMPONENTS attribute lists the allowed modules or modular devices and the constraints for their use.

Lexical structure

COMPONENTS [component-reference<exp>, AUTO_CREATE, MINIMUM_NUMBER, MAXIMUM_NUMBER, FILTER, REQUIRED_RANGES] +

The attributes of COMPONENTS are specified in Table 74.

Table 74 – COMPONENTS attribute

Usage	Attribute	Description
m	component-reference	A component in the relationship.
o	AUTO_CREATE	Purpose The number of automatically created components. Lexical structure AUTO_CREATE (expression)<exp>
o	FILTER	Purpose Used if the reference refers to a COMPONENT_FOLDER and specifies a boolean expression that filters the components. Components are included if the expression is TRUE. To filter special COMPONENTS, references to component type are used in the expression. The condition is evaluated for each type contained in the COMPONENT_FOLDER. Lexical structure FILTER (TRUE, FALSE)<exp>

Usage	Attribute	Description
o	MAXIMUM_NUMBER	Purpose The maximum allowed number of this component. Lexical structure MAXIMUM_NUMBER (expression)<exp>
o	MINIMUM_NUMBER	Purpose The minimum required number of this component. Lexical structure MINIMUM_NUMBER (expression)<exp>
o	REQUIRED_RANGES	Purpose Variables with ranges. In a valid configuration the variables have to be in the specified range. Typical this is used to define restrictions for the addressing (e.g. slot). Lexical structure REQUIRED_RANGES [reference, [MAX_VALUE, m]*, [MIN_VALUE, n]*]+

7.11.2.2 RELATION_TYPE

Purpose

The RELATION_TYPE attribute defines the relation between this component and others (child, parent or sibling).

Lexical structure

RELATION_TYPE [CHILD_COMPONENT, PARENT_COMPONENT, SIBLING_COMPONENT]

The attributes of RELATION_TYPE are specified in Table 75.

Table 75 – RELATION_TYPE attribute

Usage	Attribute	Description
s	CHILD_COMPONENT	The list of components that are allowed sub-modules.
s	PARENT_COMPONENT	The list of components that are allowed parent modules or modular devices for this component.
s	SIBLING_COMPONENT	A relation between siblings that can require or exclude other components. Requiring is possible with MINIMUM_NUMBER >0. Exclusion is possible with MAXIMUM_NUMBER = 0.
s	NEXT_COMPONENT	A special sibling relation to the next component related to the ADDRESSING. NEXT_COMPONENT is only allowed to use if the ADDRESSING is specified. Not applicable for HART.
s	PREV_COMPONENT	A special sibling relation to the previous component related to the ADDRESSING. PREV_COMPONENT is only allowed to use if the ADDRESSING is specified. Not applicable for HART.

7.11.2.3 ADDRESSING

Purpose

The ADDRESSING attribute specifies a list of variables that addresses a component uniquely.

Lexical structure

ADDRESSING variable-reference+

The attributes of ADDRESSING are specified in Table 76.

Table 76 – ADDRESSING attribute

Usage	Attribute	Description
m	variable-reference	A reference to a variable.

7.11.2.4 MAXIMUM_NUMBER

Purpose

The MAXIMUM_NUMBER attribute specifies the number of components that are allowed. The MAXIMUM_NUMBER defaults to 1.

Lexical structure

MAXIMUM_NUMBER (integer)<exp>

The attributes of MAXIMUM_NUMBER are specified in Table 77.

Table 77 – MAXIMUM_NUMBER attribute

Usage	Attribute	Description
o	integer	The total maximum allowed number of all listed components.

7.11.2.5 MINIMUM_NUMBER

Purpose

The MINIMUM_NUMBER attribute specifies the number of components that are required. The MINIMUM_NUMBER defaults to 0.

Lexical structure

MINIMUM_NUMBER (integer)<exp>

The attributes of MINIMUM_NUMBER are specified in Table 78.

Table 78 – MINIMUM_NUMBER attribute

Usage	Attribute	Description
o	integer	The total minimum required number of all listed components.

7.11.2.6 REQUIRED_INTERFACE

Purpose

The REQUIRED_INTERFACE attribute lists the required interfaces of the associated component.

Lexical structure

REQUIRED_INTERFACE interface-reference+

The attributes of REQUIRED_INTERFACE are specified in Table 79.

Table 79 – REQUIRED_INTERFACE attribute

Usage	Attribute	Description
m	interface-reference	A reference to an interface.

7.12 CONNECTION

7.12.1 General structure

Purpose

A device may include multiple EDD applications. Each EDD application is represented as an EDD application process instance. The basic construct CONNECTION provides a reference to an individual EDD application process instance within a device.

The name associated with a CONNECTION construct may be used within the COMMAND attribute CONNECTION.

Lexical structure

CONNECTION identifier, APPINSTANCE

The attribute of CONNECTION is specified in Table 80.

Table 80 – CONNECTION attribute

Usage	Attribute	Description
m	APPINSTANCE	Lexical element (see 7.12.2).

7.12.2 Specific attribute – APPINSTANCE

Purpose

The APPINSTANCE attribute specifies a EDD application process instance. It is possible to define multiple EDD applications in a device. Each EDD application represents an EDD application process instance. EDD application process instances allow different access levels.

Lexical structure

APPINSTANCE (integer, reference)<exp>

The attribute of APPINSTANCE is specified in Table 81.

Table 81 – APPINSTANCE attribute

Usage	Attribute	Description
s	integer	The number of the EDD application process instance.
s	reference	A reference to a VARIABLE instance containing the number of the EDD application process instances.

7.13 DOMAIN

Purpose

A DOMAIN represents a memory space in a device. It may contain programs or data. The identifier specifies the corresponding DOMAIN in the device. The data, programs (which are allocated in the DOMAIN), size, access, etc. are specified outside of the EDD.

NOTE 1 DOMAIN can be used to transfer large amounts of data to and from a device.

NOTE 2 To allow DOMAIN upload or download the device needs to support DOMAIN specific services.

Lexical structure

DOMAIN identifier [HANDLING, RESPONSE_CODES]

The attributes of DOMAIN are specified in Table 82.

Table 82 – DOMAIN attributes

Usage	Attribute	Description
o	HANDLING	Lexical element (see 7.36.7).
o	RESPONSE_CODES	Lexical element (see 7.36.13).

7.14 EDIT_DISPLAY

7.14.1 General structure

Purpose

An EDIT_DISPLAY defines how data will be managed for display and editing purposes. It is used to group items together for editing.

NOTE This construct is included for backwards compatibility. For future implementation the more general basic construct MENU is recommended.

Lexical structure

EDIT_DISPLAY identifier [EDIT_ITEMS, LABEL, DISPLAY_ITEMS, PRE_EDIT_ACTIONS, POST_EDIT_ACTIONS]

The attributes of EDIT_DISPLAY are specified in Table 83.

Table 83 – EDIT_DISPLAY attributes

Usage	Attribute	Description
m	EDIT_ITEMS	Lexical element (see 7.14.2.1).
m	LABEL	Lexical element (see 7.36.9).
o	DISPLAY_ITEMS	Lexical element (see 7.14.2.2).
o	POST_EDIT_ACTIONS	Lexical element (see 7.14.2.3).
o	PRE_EDIT_ACTIONS	Lexical element (see 7.14.2.4).

7.14.2 Specific attributes

7.14.2.1 EDIT_ITEMS

Purpose

The EDIT_ITEMS attribute defines the set of data items which shall be presented to the user, and may be edited by the user: VARIABLES, WRITE_AS_ONE relations, BLOCK_A PARAMETERS, and elements of BLOCK_A PARAMETERS (i.e. fields or elements of RECORD or VALUE_ARRAY type BLOCK_A PARAMETERS, respectively).

NOTE 1 Specific details for presentation and editing of a data item (for example, default value, scaling factor, range) are defined within the definitions of each referenced individual item.

NOTE 2 If a WRITE_AS_ONE relation appears as an EDIT_ITEM, the user should examine all the variables in the WRITE_AS_ONE relation. The user need not modify all the values but should at least validate that the current values are acceptable.

Lexical structure

EDIT_ITEMS [(reference)<exp>]+

The attribute of EDIT_ITEMS is specified in Table 84.

Table 84 – EDIT_ITEMS attribute

Usage	Attribute	Description
m	reference	A reference to a VARIABLE, a WRITE_AS_ONE, a BLOCK_A PARAMETERS instance or an element of BLOCK_A PARAMETERS. The allowed references are profile specific.

7.14.2.2 DISPLAY_ITEMS

Purpose

The DISPLAY_ITEMS attribute defines the set of data item which shall be presented to the user for informational purposes, i.e., they shall not be edited: VARIABLES, BLOCK_A PARAMETERS, and elements of BLOCK_A PARAMETERS (i.e. fields or elements of RECORD or VALUE_ARRAY type BLOCK_A PARAMETERS, respectively).

NOTE Specific details for presentation of a data item (for example, default value, scaling factor, range) are defined within the definitions of each referenced individual item.

Lexical structure

DISPLAY_ITEMS [(reference)<exp>]+

The attribute of DISPLAY_ITEM is specified in Table 85.

Table 85 – DISPLAY_ITEM attributes

Usage	Attribute	Description
m	reference	A reference to a VARIABLE, a WRITE_AS_ONE, a BLOCK_A PARAMETERS instance or an element of BLOCK_A PARAMETERS. The allowed references are profile specific.

7.14.2.3 POST_EDIT_ACTIONS

Purpose

The POST_EDIT_ACTIONS attribute specifies METHODS that shall be executed after the user has finished processing the EDIT_DISPLAY. If the EDIT_DISPLAY is aborted, POST_EDIT_ACTIONS are not executed. The specified METHODS shall be executed in the order they appear. If a METHOD exits abnormally, the following METHODS are not executed.

Any POST_EDIT_ACTIONS of a referenced entity in the EDIT_ITEMS shall be executed after a new value is entered. The POST_EDIT_ACTIONS of the EDIT_DISPLAY are executed only after all these individual POST_EDIT_ACTIONS have been completed.

The POST_EDIT_ACTIONS attribute shall not be expressed using a conditional IF, IF-ELSE, SELECT. If conditional behavior is required, the methods invoked by the action shall accommodate it.

Lexical structure

POST_EDIT_ACTIONS [reference]+, DEFINITION

The attributes of POST_EDIT_ACTIONS are specified in Table 86.

7.14.2.4 PRE_EDIT_ACTIONS

Purpose

The PRE_EDIT_ACTIONS attribute specifies METHODS that shall be executed immediately when the EDIT_DISPLAY is activated. The specified METHODS shall be executed in the order they appear. If a METHOD exits abnormally, the following METHODS are not executed and EDIT_DISPLAY activation is cancelled.

The PRE_EDIT_ACTIONS of the EDIT_DISPLAY shall be executed before any of the defined PRE_EDIT_ACTIONS of the referenced entities.

The PRE_EDIT_ACTIONS attribute shall not be expressed using a conditional (IF, IF-ELSE, SELECT). If conditional behavior is required, the methods invoked by the action shall accommodate it.

NOTE PRE_EDIT_ACTIONS of all associated VARIABLE objects are executed immediately before the user actually starts to edit them for example by setting a cursor in an edit field. Displaying VARIABLE objects without editing does not fire PRE_EDIT_ACTIONS of VARIABLE objects.

Lexical structure

PRE_EDIT_ACTIONS [reference]+, DEFINITION

The attributes of PRE_EDIT_ACTIONS are specified in Table 86.

Table 86 – POST_EDIT_ACTIONS, PRE_EDIT_ACTIONS attribute

Usage	Attribute	Description
m	reference	A reference to a METHOD instance or a METHOD instance with arguments.
o	DEFINITION	Lexical element (see 7.36.1).

7.15 FILE

Purpose

FILE describes an area of persistent storage that is maintained by the EDD application. The store shall only contain data. Each physical device has its own copy of the store, i.e., a FILE associated with an instance of a device, not with the type of a device. The MEMBERS attribute defines the structure of the store, which shall be fixed, that means the MEMBERS attribute shall not be expressed using a conditional IF, IF-ELSE, SELECT.

Lexical structure

FILE identifier [MEMBERS, HELP, LABEL]

The attributes of FILE are specified in Table 87.

Table 87 – FILE attributes

Usage	Attribute	Description
m	MEMBERS	Lexical element (see 7.36.12).
o	HELP	Lexical element (see 7.36.8).
o	LABEL	Lexical element (see 7.36.9).

7.16 GRAPH

7.16.1 General structure

Purpose

GRAPH describes a graph used to display data from a device in the EDD application. A GRAPH is used to display a data set that is stored in a device, as opposed to a CHART that is used to display continuously collected data values from a device.

Lexical structure

GRAPH identifier [MEMBERS, CYCLE_TIME, HEIGHT, HELP, LABEL, VALIDITY, WIDTH, X_AXIS]

The attributes of GRAPH are specified in Table 88.

Table 88 – GRAPH attributes

Usage	Attribute	Description
m	MEMBERS	Lexical element (see 7.36.12).
o	CYCLE_TIME	Lexical element (see 7.16.2.1).
o	HEIGHT	Lexical element (see 7.36.5).
o	HELP	Lexical element (see 7.36.8).
o	LABEL	Lexical element (see 7.36.9).
o	VALIDITY	Lexical element (see 7.38.16).
o	WIDTH	Lexical element (see 7.36.15).
o	X_AXIS	Lexical element (see 7.16.2.2).

7.16.2 Specific attributes

7.16.2.1 CYCLE_TIME

Purpose

The CYCLE_TIME attribute specifies the length of time, in ms, the EDD application shall wait between successive reads of the source values in the device. By default, a GRAPH has no cycle time.

Lexical structure

CYCLE_TIME (integer)<exp>

The attribute of CYCLE_TIME is specified in Table 89.

Table 89 – CYCLE_TIME attribute

Usage	Attribute	Description
m	integer	The length of time, in ms, between successive reads of the source values in the device.

7.16.2.2 X_AXIS

Purpose

The X_AXIS attribute specifies the way in which the X_AXIS is displayed. Each WAVEFORM on a GRAPH shares a common X_AXIS but each WAVEFORM on a GRAPH has its own Y_AXIS. By default, the display of the X axis of a GRAPH without an X_AXIS attribute is inferred from the WAVEFORMs on the GRAPH.

Lexical structure

X_AXIS (reference)<exp>

The attribute of X_AXIS is specified in Table 90.

Table 90 – X_AXIS attribute

Usage	Attribute	Description
m	reference	A reference to an AXIS instance.

7.17 GRID

7.17.1 General structure

Purpose

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.

Lexical structure

GRID identifier [VECTORS, HANDLING, HEIGHT, HELP, LABEL, ORIENTATION, VALIDITY, WIDTH]

The attributes of GRID are specified in Table 91.

Table 91 – GRID attributes

Usage	Attribute	Description
m	VECTORS	Lexical element (see 7.17.2.1).
o	HANDLING	Lexical element (see 7.36.6).
o	HEIGHT	Lexical element (see 7.36.5).
o	HELP	Lexical element (see 7.36.8).
o	LABEL	Lexical element (see 7.36.9).
o	ORIENTATION	Lexical element (see 7.17.2.2).
o	VALIDITY	Lexical element (see 7.36.16).
o	WIDTH	Lexical element (see 7.36.17).

7.17.2 Specific attributes

7.17.2.1 VECTORS

Purpose

The VECTORS attribute specifies the contents of the GRID. Each VECTOR is displayed in order. For a vertical GRID, the vectors are displayed left to right. For a horizontal GRID, the vectors are displayed top to bottom. Each vector consists of a description that provides a label for the vector followed by the data to be displayed.

Lexical structure

```
VECTORS [description, (reference, string, double, float, integer,
unsigned_integer)<exp>+]
```

The attributes of VECTORS are specified in Table 92.

Table 92 – VECTORS attribute

Usage	Attribute	Description
m	description	A string containing a short description of the data within the vector.
o	reference	A reference to either a VARIABLE instance, a member of a RECORD instance, an element of a VALUE_ARRAY instance, an element of a REFERENCE_ARRAY instance, a VALUE_ARRAY instance or a REFERENCE_ARRAY instance.
	string	A string constant or a reference that resolves to a string.
o	double	A double precision floating point constant.
o	float	A single precision floating point constant.
o	integer	An integer constant.
o	unsigned_integer	An unsigned integer constant.

7.17.2.2 ORIENTATION

Purpose

The ORIENTATION attribute specifies the direction the vectors are displayed. By default, a GRID without a ORIENTATION attribute is displayed vertically.

Lexical structure

```
ORIENTATION (HORIZONTAL, VERTICAL) <exp>
```

The attributes of ORIENTATION are specified in Table 93.

Table 93 – ORIENTATION attribute

Usage	Attribute	Description
s	HORIZONTAL	The vectors are to be displayed in rows.
s	VERTICAL	The vectors are to be displayed in columns.

7.18 IMAGE

7.18.1 General structure

Purpose

IMAGE describes a visual image.

Lexical structure

```
IMAGE identifier [PATH, HELP, LABEL, LINK, VALIDITY]
```

The attributes of IMAGE are specified in Table 94.

Table 94 – IMAGE attributes

Usage	Attribute	Description
m	PATH	Lexical element (see 7.18.2.1).
o	HELP	Lexical element (see 7.36.8).
o	LABEL	Lexical element (see 7.36.9).
o	LINK	Lexical element (see 7.18.2.2).
o	VALIDITY	Lexical element (see 7.36.16).

7.18.2 Specific attributes

7.18.2.1 PATH

Purpose

The PATH attribute specifies the filename of the image. Images for multiple locales may be specified using the same localization technique used with string literals. The ZZ country code may be used to specify a low resolution version of the image, which are useful for EDD applications that run on mobile devices having a limited memory.

EXAMPLE "en|english.jpeg|de|german.jpeg|en ZZ|english-lowres.jpeg|de ZZ|german-lowres.jpeg"

Lexical structure

PATH (file-name)<exp>

The attributes of PATH are specified in Table 95.

Table 95 – PATH attribute

Usage	Attribute	Description
m	file-name	A string literal that identifies the filename of the image.

7.18.2.2 LINK

Purpose

The LINK attribute specifies a MENU or a METHOD that can be accessed from the IMAGE. This allows additional information to be provided about the IMAGE.

Lexical structure

LINK (reference)<exp>

The attributes of LINK are specified in Table 96.

Table 96 – LINK attribute

Usage	Attribute	Description
m	reference	A reference to either a MENU instance or a METHOD instance.

7.19 IMPORT

7.19.1 General structure

Purpose

The EDDL constructs previously specified in this clause are sufficient for describing any single device. However, additional mechanisms are required to describe multiple revisions of a single device or to describe parts of a generic device. To provide these mechanisms, one EDD can import another EDD. The imported EDD itself can also import other EDDs.

NOTE With these mechanisms the EDD of a new device revision can be specified by simply importing the EDD of the old device revision and specifying the changes of the items. This type of EDD is called delta description, because the entire EDD is specified as changes to an existing EDD.

Three import types shall be supported:

- importing all items, where all items of the external EDD are imported;
- importing items of a specified type, where only the items of the specified types are imported. If a specified type is imported, further imports of specific items of the same type are not allowed;
- importing a specific item.

The identifier of an imported EDD element shall not be changed. Figure 27 shows the EDDL import mechanisms.

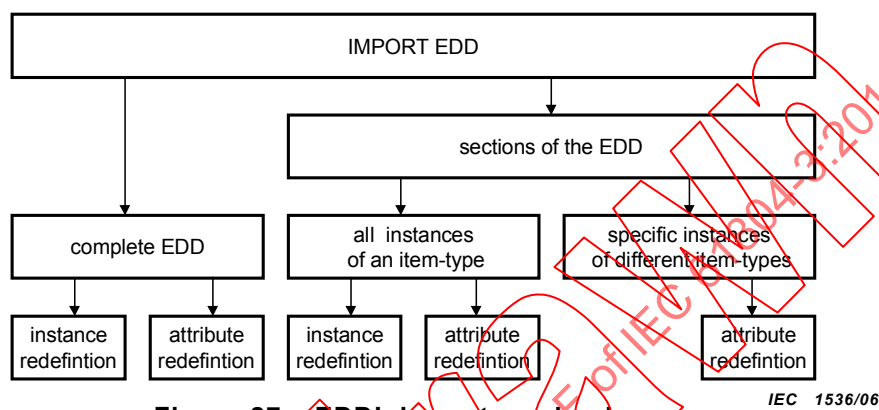


Figure 27 – EDDL import mechanisms

An alternative way to import is to reference the EDD by file name.

Lexical structure

```

IMPORT ((DD_REVISION, DEVICE_REVISION, DEVICE_TYPE, EDD_PROFILE,
EDD_VERSION, MANUFACTURER, MANUFACTURER_EXT), FILE_NAME), (EVERYTHING,
(AXES, BLOCKS_A, BLOCKS_B, CHART, COLLECTIONS, COMMANDS, COMPONENTS,
COMPONENT_FOLDERS, COMPONENT_REFERENCES, COMPONENT_RELATIONS, CONNECTIONS,
DOMAINS, EDIT_DISplays, FILES, GRAPHS, GRIDS, IMAGES, INTERFACES, LISTS,
MENUS, METHODS, PROGRAMS, RECORDS, REFERENCE_ARRAYS, REFRESHES, RELATIONS,
RESPONSE_CODES, SOURCES, TEMPLATES, UNITS, VALUE_ARRAYS, VARIABLES,
VARIABLE_LISTS, WAVEFORMS, WRITE_AS_ONES, reference*,
[DELETE identifier]*,
[REDEFINE identifier],
attribute-redefinition*))
  
```

The attributes are specified in Table 97.

Table 97 – Importing Device Description

Usage	Attribute	Description
o	attribute-redefinition	See 7.19.2.
o	AXES	All AXIS instances of an external EDD are imported.
o	BLOCKS_A	All BLOCK_A instances of an external EDD are imported.
o	BLOCKS_B	All BLOCK_B instances of an external EDD are imported.
o	CHARTS	All CHART instances of an external EDD are imported.
o	COLLECTIONS	All COLLECTION instances of an external EDD are imported.
o	COMMANDS	All COMMAND instances of an external EDD are imported.
o	COMPONENTS	All COMPONENT instances of an external EDD are imported.
o	COMPONENT_FOLDERS	All COMPONENT_FOLDER instances of an external EDD are imported.
o	COMPONENT_REFERENCES	All COMPONENT_REFERENCE instances of an external EDD are imported.
o	COMPONENT_RELATIONS	All COMPONENT_RELATION instances of an external EDD are imported.
o	CONNECTIONS	All CONNECTION instances of an external EDD are imported.

Usage	Attribute	Description
m	DD_REVISION	See 7.2.2.1.
o	DELETE	An imported instance shall be deleted.
m	DEVICE_REVISION	See 7.2.2.2.
m	DEVICE_TYPE	See 7.2.2.3.
o	DOMAINS	All DOMAIN instances of an external EDD are imported.
o	EDD_PROFILE	See 7.2.2.4.
o	EDD_VERSION	See 7.2.2.5.
o	EDIT_DISPLAYS	All EDIT_DISPLAY instances of an external EDD are imported.
o	EVERYTHING	All items of an external EDD are imported. If this form is used, no item types and no single instances can be selected. The attributes DELETE, REDEFINE and attribute-redefinition may be used.
o	FILE_NAME	A string which contains a file name and its path in a directory structure. The file contains the EDD which shall be imported. Using this kind of referencing, the specification of the EDD identification information is not necessary.
o	FILES	All FILE instances of an external EDD are imported.
o	GRAPHS	All GRAPH instances of an external EDD are imported.
o	GRIDS	All GRID instances of an external EDD are imported.
o	IMAGES	All IMAGE instances of an external EDD are imported.
o	INTERFACES	All INTERFACE instances of an external EDD are imported.
o	LISTS	All LIST instances of an external EDD are imported.
m	MANUFACTURER	See 7.2.2.6.
c	MANUFACTURER_EXT	See 7.2.2.7.
o	MENUS	All MENU instances of an external EDD are imported.
o	METHODS	All METHOD instances of an external EDD are imported.
o	PROGRAMS	All PROGRAM instances of an external EDD are imported.
o	RECORDS	All RECORD instances of an external EDD are imported.
o	REDEFINE	All attributes of an imported instance shall be redefined.
o	reference	A single instance is imported. A single instance shall not be imported, if it is already included in an item type import.
o	REFERENCE_ARRAYS	All REFERENCE_ARRAY instances of an external EDD are imported.
o	REFRESHES	All REFRESH instances of an external EDD are imported.
o	RELATIONS	All REFRESH, UNIT and WRITE_AS_ONE instances of an external EDD are imported.
o	RESPONSE_CODES	All RESPONSE_CODE instances of an external EDD are imported.
o	SOURCES	All SOURCE instances of an external EDD are imported.
o	TEMPLATES	All TEMPLATE instances of an external EDD are imported.
o	UNITS	All UNIT instances of an external EDD are imported.
o	VALUE_ARRAYS	All VALUE_ARRAY instances of an external EDD are imported.
o	VARIABLE_LISTS	All VARIABLE_LIST instances of an external EDD are imported.
o	VARIABLES	All VARIABLE instances of an external EDD are imported.
o	WAVEFORMS	All WAVEFORM instances of an external EDD are imported.
o	WRITE_AS_ONES	All WRITE_AS_ONE instances of an external EDD are imported.

7.19.2 Specific attributes

7.19.2.1 Attribute redefinition

Purpose

Attributes of an imported instance may be added, deleted or redefined. Adding, deleting and redefining do not affect all attributes of an EDD basic construct element.

The redefinition attributes are specified in Table 98.

Table 98 – Redefinition attributes

Usage	Attribute	Description
s	ADD	Adds the following specified attribute. "A" indicates that ADD can be applied to the attribute in the redefinition rules tables (see Table 99 to Table 129).
s	DELETE	Deletes the following specified attribute. "D" indicates that DELETE can be applied to the attribute in the redefinition rules tables (see Table 99 to Table 129).
s	REDEFINE	Redefines the following specified attribute. "R" indicates that REDEFINE can be applied to the attribute in the redefinition rules tables (see Table 99 to Table 129).

Lexical structure – AXIS

AXIS identifier, [[DELETE, REDEFINE], [CONSTANT_UNIT, HELP, LABEL, MAX_VALUE, MIN_VALUE, SCALING]]+

The redefinition rules of AXIS are specified in Table 98 and Table 99.

Table 99 – Redefinition rules for AXIS attributes

A	D	R	Attribute	Description
	•	•	CONSTANT_UNIT	Lexical element (see 7.3.2.1).
	•	•	HELP	Lexical element (see 7.36.8).
	•	•	LABEL	Lexical element (see 7.36.9).
	•	•	MAX_VALUE	Lexical element (see 7.3.2.2).
	•	•	MIN_VALUE	Lexical element (see 7.3.2.3).
	•	•	SCALING	Lexical element (see 7.3.2.4).

Lexical structure – BLOCK_A

BLOCK identifier, [[ADD, DELETE, REDEFINE], [CHARACTERISTICS, LABEL, PARAMETERS, parameters-list-element, AXIS_ITEMS, CHART_ITEMS, COLLECTION_ITEMS, EDIT_DISPLAY_ITEMS, FILE_ITEMS, GRAPH_ITEMS, GRID_ITEMS, HELP, IMAGE_ITEMS, LIST_ITEMS, LOCAL_PARAMETERS, local-parameters-list-element, MENU_ITEMS, METHOD_ITEMS, PARAMETER_LISTS, parameter-lists-list-element, REFERENCE_ARRAY_ITEMS, REFRESH_ITEMS, SOURCE_ITEMS, UNIT_ITEMS, WAVEFORM_ITEMS, WRITE_AS_ONE_ITEMS, CHARTS, charts-element, LISTS, lists-element, GRAPHS, graphs-element, GRIDS, grids-element, MENUS, menus-element, METHODS, methods-element]]+

The redefinition rules of BLOCK_A are specified in Table 98 and Table 100.

Table 100 – Redefinition rules for BLOCK_A attributes

A	D	R	Attribute	Description
		•	CHARACTERISTICS	Lexical element (see 7.4.1.2.1).
		•	LABEL	Lexical element (see 7.36.9).
		•	PARAMETERS	Lexical element (see 7.4.1.2.2).
•		•	parameters-list-element	Single enumeration list elements of PARAMETERS may be changed.
	•	•	AXIS_ITEMS	Lexical element (see 7.4.1.2.3).
	•	•	CHART_ITEMS	Lexical element (see 7.4.1.2.4).
	•	•	COLLECTION_ITEMS	Lexical element (see 7.4.1.2.5).
	•	•	EDIT_DISPLAY_ITEMS	Lexical element (see 7.4.1.2.6).
	•	•	FILE_ITEMS	Lexical element (see 7.4.1.2.7).

A	D	R	Attribute	Description
	•	•	GRAPH_ITEMS	Lexical element (see 7.4.1.2.8).
	•	•	GRID_ITEMS	Lexical elements (see 7.4.1.2.9).
	•	•	HELP	Lexical element (see 7.36.8).
	•	•	IMAGE_ITEMS	Lexical elements (see 7.4.1.2.10).
	•	•	LIST_ITEMS	Lexical element (see 7.4.1.2.11).
		•	LOCAL_PARAMETERS	Lexical element (see 7.4.1.2.12).
•		•	local-parameters-list-element	single enumeration list elements of LOCAL_PARAMETERS may be changed.
	•	•	MENU_ITEMS	Lexical element (see 7.4.1.2.13).
	•	•	METHOD_ITEMS	Lexical element (see 7.4.1.2.14).
		•	PARAMETER_LISTS	Lexical element (see 7.4.1.2.15).
•		•	parameter-lists-list-element	Single enumeration list elements of PARAMETERS_LISTS may be changed.
	•	•	REFERENCE_ARRAY_ITEMS	Lexical element (see 7.4.1.2.16).
	•	•	REFRESH_ITEMS	Lexical element (see 7.4.1.2.17).
	•	•	SOURCE_ITEMS	Lexical element (see 7.4.1.2.18).
	•	•	UNIT_ITEMS	Lexical element (see 7.4.1.2.19).
	•	•	WAVEFORM_ITEMS	Lexical element (see 7.4.1.2.20).
	•	•	WRITE_AS_ONE_ITEMS	Lexical element (see 7.4.1.2.21).
	•	•	CHARTS	Lexical element (see 7.4.1.2.22).
•	•	•	charts-element	Single elements of CHARTS may be changed.
	•	•	LISTS	Lexical element (see 7.4.1.2.23).
•	•	•	lists-element	Single elements of LISTS may be changed.
	•	•	GRAPHS	Lexical element (see 7.4.1.2.24).
•	•	•	graphs-element	Single elements of GRAPHS may be changed.
	•	•	GRIDS	Lexical element (see 7.4.1.2.25).
•	•	•	grids-element	Single elements of GRIDS may be changed.
	•	•	MENUS	Lexical element (see 7.4.1.2.26).
•	•	•	menus-element	Single elements of MENUS may be changed.
	•	•	METHODS	Lexical element (see 7.4.1.2.27).
•	•	•	methods-element	Single elements of METHODS may be changed.

Lexical structure – BLOCK_B

BLOCK identifier, [REDEFINE, [NUMBER, TYPE]]+

The redefinition rules of BLOCK_B are specified in Table 98 and Table 101.

Table 101 – Redefinition rules for BLOCK_B attributes

A	D	R	Attribute	Description
		•	NUMBER	Lexical element (see 7.4.2.2.1).
		•	TYPE	Lexical element (see 7.4.2.2.2).

Lexical structure — CHART

CHART identifier, [[ADD, DELETE, REDEFINE], [MEMBERS, members-list-element, CYCLE_TIME, HEIGHT, HELP, LABEL, LENGTH, TYPE, VALIDITY, WIDTH]]+

The redefinition rules of CHART are specified in Table 98 and Table 102.

Table 102 – Redefinition rules for CHART attributes

A	D	R	Attribute	Description
		•	MEMBERS	Lexical element (see 7.36.12).
•	•	•	members-list-element	Single enumeration list elements of MEMBERS may be changed.
	•	•	CYCLE_TIME	Lexical element (see 7.5.2.1).
	•	•	HEIGHT	Lexical element (see 7.36.7).
	•	•	HELP	Lexical element (see 7.36.8).
	•	•	LABEL	Lexical element (see 7.36.9).
	•	•	LENGTH	Lexical element (see 7.5.2.2).
	•	•	TYPE	Lexical element (see 7.5.2.3).
	•	•	VALIDITY	Lexical element (see 7.36.16).
	•	•	WIDTH	Lexical element (see 7.36.15).

Lexical structure — COLLECTION

COLLECTION identifier, [[ADD, DELETE, REDEFINE], [MEMBERS, members-list-element, HELP, LABEL, VALIDITY]]+

The redefinition rules of COLLECTION are specified in Table 98 and Table 103.

Table 103 – Redefinition rules for COLLECTION attributes

A	D	R	Attribute	Description
		•	MEMBERS	Lexical element (see 7.36.12).
•	•	•	members-list-element	Single enumeration list elements of MEMBERS may be changed.
	•	•	HELP	Lexical element (see 7.36.8).
	•	•	LABEL	Lexical element (see 7.36.9).
	•	•	VALIDITY	Lexical element (see 7.36.16).

Lexical structure — COMMAND

COMMAND identifier, [[ADD, DELETE, REDEFINE], [OPERATION, TRANSACTION, INDEX, BLOCK_B, NUMBER, SLOT, SUB_SLOT, CONNECTION, HEADER, MODULE, RESPONSE_CODES, response-codes-list-element]]+

The redefinition rules of COMMAND are specified in Table 98 and Table 104.

Table 104 – Redefinition rules for COMMAND attributes

A	D	R	Attribute	Description
			OPERATION	Lexical element (see 7.7.2.1).
•	•	•	TRANSACTION	Lexical element (see 7.7.2.2).
	•	•	INDEX	Lexical element (see 7.7.2.3).
			BLOCK_B	Lexical element (see 7.7.2.4).
			NUMBER	Lexical element (see 7.7.2.5).
	•	•	SLOT	Lexical element (see 7.7.2.6).
	•	•	SUB_SLOT	Lexical element (see 7.7.2.7).
	•	•	CONNECTION	Lexical element (see 7.7.2.7).
	•	•	HEADER	Lexical element (see 7.7.2.9).
	•	•	MODULE	Lexical element (see 7.7.2.10).
	•	•	RESPONSE_CODES	Lexical element (see 7.36.13).
•	•	•	response-codes-list-element	Single enumeration list elements of RESPONSE_CODE may be changed.

Lexical structure — COMPONENT

COMPONENT identifier, [[DELETE, REDEFINE], [LABEL, CAN_DELETE, CHECK_CONFIGURATION, CLASSIFICATION, HELP, COMPONENT_PARENT, COMPONENT_PATH, COMPONENT_RELATIONS, DECLARATION, DETECT, EDD, INITIAL_VALUES, PROTOCOL, REDUNDANCY, SCAN, SCAN_LIST, SUPPLIED_INTERFACE]]+

The redefinition rules of COMPONENT are specified in Table 98 and Table 105.

Table 105 – Redefinition rules for COMPONENT attributes

A	D	R	Attribute	Description
		•	LABEL	Lexical element (see 7.36.9).
	•	•	CAN_DELETE	Lexical element (see 7.8.2.1).
	•	•	CHECK_CONFIGURATION	Lexical element (see 7.8.2.2).
	•	•	CLASSIFICATION	Lexical element (see 7.36.1).
	•	•	COMPONENT_PARENT	Lexical element (see 7.36.2).
	•	•	COMPONENT_PATH	Lexical element (see 7.36.3).
	•	•	COMPONENT_RELATIONS	Lexical element (see 7.8.2.3).
	•	•	DECLARATION	Lexical element (see 7.8.2.4).
	•	•	DETECT	Lexical element (see 7.8.2.5).
	•	•	EDD	Lexical element (see 7.8.2.6).
	•	•	HELP	Lexical element (see 7.36.8).
	•	•	INITIAL_VALUES	Lexical element (see 7.8.2.7).
	•	•	PROTOCOL	Lexical element (see 7.36.13).
	•	•	REDUNDANCY	Lexical element (see 7.8.2.8).
	•	•	SCAN	Lexical element (see 7.8.2.9).
	•	•	SCAN_LIST	Lexical element (see 7.8.2.10).
	•	•	SUPPLIED_INTERFACE	Lexical element (see 7.36.15).

Lexical structure — COMPONENT_FOLDER

COMPONENT_FOLDER identifier, [[DELETE, REDEFINE], [LABEL, CLASSIFICATION, COMPONENT_PARENT, COMPONENT_PATH, HELP, PROTOCOL]]+

The redefinition rules of COMPONENT_FOLDER are specified in Table 98 and Table 106.

Table 106 – Redefinition rules for COMPONENT_FOLDER attributes

A	D	R	Attribute	Description
		•	LABEL	Lexical element (see 7.36.9).
	•	•	CLASSIFICATION	Lexical element (see 7.36.1).
	•	•	COMPONENT_PARENT	Lexical element (see 7.36.2).
	•	•	COMPONENT_PATH	Lexical element (see 7.36.3).
	•	•	HELP	Lexical element (see 7.36.8).
	•	•	PROTOCOL	Lexical element (see 7.36.13).

Lexical structure — COMPONENT_REFERENCE

COMPONENT_REFERENCE identifier, [[DELETE, REDEFINE], [CLASSIFICATION, COMPONENT_PARENT, COMPONENT_PATH, DEVICE_REVISION, DEVICE_TYPE, MANUFACTURER, PROTOCOL]]+

The redefinition rules of COMPONENT_REFERENCE are specified in Table 98 and Table 107.

Table 107 – Redefinition rules for COMPONENT_REFERENCE attributes

A	D	R	Attribute	Description
	•	•	CLASSIFICATION	Lexical element (see 7.36.1).
	•	•	COMPONENT_PARENT	Lexical element (see 7.36.2).
	•	•	COMPONENT_PATH	Lexical element (see 7.36.3).
	•	•	DEVICE_REVISION	Lexical element (see 7.2.2.2).
	•	•	DEVICE_TYPE	Lexical element (see 7.2.2.3).
	•	•	MANUFACTURER	Lexical element (see 7.2.2.6).
	•	•	PROTOCOL	Lexical element (see 7.36.13).

Lexical structure — COMPONENT_RELATION

COMPONENT_RELATION identifier, [[ADD, DELETE, REDEFINE], [COMPONENTS, AUTO_CREATE, FILTER, REQUIRED_RANGES, LABEL, RELATION_TYPE, ADDRESSING, HELP, MAXIMUM_NUMBER, MINIMUM_NUMBER, REQUIRED_INTERFACE, SUPPLIED_INTERFACE]]+

The redefinition rules of COMPONENT_RELATION are specified in Table 98 and Table 108.

Table 108 – Redefinition rules for COMPONENT_RELATION attributes

A	D	R	Attribute	Description
		•	COMPONENTS	Lexical element (see 7.11.2.1).
•	•	•	components-list-element	Single list elements of COMPONENTS may be changed.
	•	•	AUTO_CREATE	Lexical element (see 7.11.2.1).
	•	•	FILTER	Lexical element (see 7.11.2.1).
	•	•	REQUIRED_RANGES	Lexical element (see 7.11.2.1).
		•	LABEL	Lexical element (see 7.36.9).
		•	RELATION_TYPE	Lexical element (see 7.11.2.2).
	•	•	ADDRESSING	Lexical element (see 7.11.2.3).
	•	•	HELP	Lexical element (see 7.36.8).
	•	•	MAXIMUM_NUMBER	Lexical element (see 7.11.2.4).
	•	•	MINIMUM_NUMBER	Lexical element (see 7.11.2.5).
	•	•	REQUIRED_INTERFACE	Lexical element (see 7.11.2.6).
	•	•	SUPPLIED_INTERFACE	Lexical element (see 7.36.15).

Lexical structure — CONNECTION

CONNECTION identifier, [REDEFINE, [APPINSTANCE]]+

The redefinition rules of CONNECTION are specified in Table 98 and Table 109.

Table 109 – Redefinition rules for CONNECTION attributes

A	D	R	Attribute	Description
		•	APPINSTANCE	Lexical element (see 7.12.2).

Lexical structure — DOMAIN

DOMAIN identifier, [DELETE, REDEFINE], [HANDLING, RESPONSE_CODES]]+

The redefinition rules of DOMAIN are specified in Table 98 and Table 110.

Table 110 – Redefinition rules for DOMAIN attributes

A	D	R	Attribute	Description
	•	•	HANDLING	Lexical element (see 7.36.6).
	•	•	RESPONSE_CODES	Lexical element (see 7.36.13).

Lexical structure — EDIT_DISPLAY

EDIT_DISPLAY identifier, [[DELETE, REDEFINE], [EDIT_ITEMS, LABEL, DISPLAY_ITEMS, POST_EDIT_ACTIONS, PRE_EDIT_ACTIONS]]+

The redefinition rules of EDIT_DISPLAY are specified in Table 98 and Table 111.

Table 111 – Redefinition rules for EDIT_DISPLAY attributes

A	D	R	Attribute	Description
		•	EDIT_ITEMS	Lexical element (see 7.14.2.1).
		•	LABEL	Lexical element (see 7.36.9).
	•	•	DISPLAY_ITEMS	Lexical element (see 7.14.2.2).
	•	•	POST_EDIT_ACTIONS	Lexical element (see 7.14.2.3).
	•	•	PRE_EDIT_ACTIONS	Lexical element (see 7.14.2.4).

Lexical structure — FILE

FILE identifier, [[ADD, DELETE, REDEFINE], [MEMBERS, members-list-element, HELP, LABEL]]+

The redefinition rules of FILE are specified in Table 98 and Table 112.

Table 112 – Redefinition rules for FILE attributes

A	D	R	Attribute	Description
		•	MEMBERS	Lexical element (see 7.36.12).
•	•	•	members-list-element	Single enumeration list elements of MEMBERS may be changed.
	•	•	HELP	Lexical element (see 7.36.8).
	•	•	LABEL	Lexical element (see 7.36.9).

Lexical structure — GRAPH

GRAPH identifier, [[ADD, DELETE, REDEFINE], [MEMBERS, members-list-element, CYCLE_TIME, HEIGHT, HELP, LABEL, VALIDITY, WIDTH, X_AXIS]]+

The redefinition rules of GRAPH are specified in Table 98 and Table 113.

Table 113 – Redefinition rules for GRAPH attributes

A	D	R	Attribute	Description
		•	MEMBERS	Lexical element (see 7.36.12).
•	•	•	members-list-element	Single enumeration list elements of MEMBERS may be changed.
	•	•	CYCLE_TIME	Lexical element (see 7.16.2.1).
	•	•	HEIGHT	Lexical element (see 7.36.7).
	•	•	HELP	Lexical element (see 7.36.8).
	•	•	LABEL	Lexical element (see 7.36.9).
	•	•	VALIDITY	Lexical element (see 7.38.16).
	•	•	WIDTH	Lexical element (see 7.36.17).
	•	•	X_AXIS	Lexical element (see 7.16.2.2).

Lexical structure — GRID

GRID identifier, [[ADD, DELETE, REDEFINE], VECTORS, HANDLING, HEIGHT, HELP, LABEL, ORIENTATION, VALIDITY, WIDTH]]+

The redefinition rules of GRAPH are specified in Table 98 and Table 114.

Table 114 – Redefinition rules for GRID attributes

A	D	R	Attribute	Description
		•	VECTORS	Lexical element (see 7.17.2.1).
	•	•	HANDLING	Lexical element (see 7.36.6).
	•	•	HEIGHT	Lexical element (see 7.36.7).
	•	•	HELP	Lexical element (see 7.36.8).
	•	•	LABEL	Lexical element (see 7.36.9).
	•	•	ORIENTATION	Lexical element (see 7.17.2.2).
	•	•	VALIDITY	Lexical element (see 7.38.16).
	•	•	WIDTH	Lexical element (see 7.36.17).

Lexical structure — IMAGE

IMAGE identifier, [[DELETE, REDEFINE], PATH, HELP, LABEL, LINK, VALIDITY]]+

The redefinition rules of GRAPH are specified in Table 98 and Table 115.

Table 115 – Redefinition rules for IMAGE attributes

A	D	R	Attribute	Description
		•	PATH	Lexical element (see 7.18.2.1).
	•	•	HELP	Lexical element (see 7.36.8).
	•	•	LABEL	Lexical element (see 7.36.9).
	•	•	LINK	Lexical element (see 7.18.2.2).
	•	•	VALIDITY	Lexical element (see 7.38.16).

Lexical structure — INTERFACE

INTERFACE identifier, [[DELETE, REDEFINE], [DECLARATION, LABEL, HELP]]+

The redefinition rules of INTERFACE are specified in Table 98 and Table 116.

Table 116 – Redefinition rules for INTERFACE attributes

A	D	R	Attribute	Description
		•	DECLARATION	Lexical element (see 7.20.2).
		•	LABEL	Lexical element (see 7.36.9).
	•	•	HELP	Lexical element (see 7.36.8).

Lexical structure — LIST

LIST identifier, [[DELETE, REDEFINE], [TYPE, CAPACITY, COUNT, HELP, LABEL, VALIDITY]]+

The redefinition rules of LIST are specified in Table 98 and Table 117.

Table 117 – Redefinition rules for LIST attributes

A	D	R	Attribute	Description
		•	TYPE	Lexical element (see 7.22.2.1).
	•	•	CAPACITY	Lexical element (see 7.22.2.2).
	•	•	COUNT	Lexical element (see 7.22.2.3).
	•	•	HELP	Lexical element (see 7.36.8).
	•	•	LABEL	Lexical element (see 7.36.9).
	•	•	VALIDITY	Lexical element (see 7.36.16).

Lexical structure — MENU

MENU identifier, [[ADD, DELETE, REDEFINE], [ITEMS, items-list-element, LABEL, ACCESS, HELP, POST_EDIT_ACTIONS, POST_READ_ACTIONS, POST_WRITE_ACTIONS, PRE_EDIT_ACTIONS, PRE_READ_ACTIONS, PRE_WRITE_ACTIONS, PURPOSE, ROLE, STYLE, VALIDITY]]+

The redefinition rules of MENU are specified in Table 98 and Table 118.

Table 118 – Redefinition rules for MENU attributes

A	D	R	Attribute	Description
		•	ITEMS	Lexical element (see 7.23.2.1).
		•	LABEL	Lexical element (see 7.36.9).
	•	•	ACCESS	Lexical element (see 7.23.2.2).
	•	•	HELP	Lexical element (see 7.36.8).
	•	•	POST_EDIT_ACTIONS	Lexical element (see 7.23.2.3).
	•	•	POST_READ_ACTIONS	Lexical element (see 7.23.2.4).
	•	•	POST_WRITE_ACTIONS	Lexical element (see 7.23.2.5).
	•	•	PRE_EDIT_ACTIONS	Lexical element (see 7.23.2.6).
	•	•	PRE_READ_ACTIONS	Lexical element (see 7.23.2.7).
	•	•	PRE_WRITE_ACTIONS	Lexical element (see 7.23.2.8).
	•	•	PURPOSE	Lexical element (see 7.23.2.9).
	•	•	ROLE	Lexical element (see 7.23.2.10).
	•	•	STYLE	Lexical element (see 7.23.2.11).
	•	•	VALIDITY	Lexical element (see 7.36.16).

Lexical structure — METHOD

METHOD identifier, [[DELETE, REDEFINE], [DEFINITION, ACCESS, CLASS, LABEL, HELP, TYPE, VALIDITY]]+

The redefinition rules of METHOD are specified in Table 98 and Table 94.

Table 119 – Redefinition rules for METHOD attributes

A	D	R	Attribute	Description
		•	DEFINITION	Lexical element (see 7.36.4).
	•	•	ACCESS	Lexical element (see 7.24.2.1).
		•	CLASS	Lexical element (see 7.24.2.2).
	•	•	LABEL	Lexical element (see 7.36.9).
	•	•	HELP	Lexical element (see 7.36.8).
			TYPE	Lexical element (see 7.24.2.3).
	•	•	VALIDITY	Lexical element (see 7.36.16).

Lexical structure — PROGRAM

PROGRAM identifier, [[DELETE, REDEFINE], [ARGUMENTS, argument-list-element, RESPONSE_CODES]]+

The redefinition rules of PROGRAM are specified in Table 98 and Table 120.

Table 120 – Redefinition rules for PROGRAM attributes

A	D	R	Attribute	Description
	•	•	ARGUMENTS	Lexical element (see 7.25.2).
	•	•	RESPONSE_CODES	Lexical element (see 7.36.13).

Lexical structure — RECORD

RECORD identifier, [[ADD, DELETE, REDEFINE], [MEMBERS, members-list-element, LABEL, HELP, RESPONSE_CODES, VALIDITY]]+

The redefinition rules of RECORD are specified in Table 98 and Table 121.

Table 121 – Redefinition rules for RECORD attributes

A	D	R	Attribute	Description
		•	MEMBERS	Lexical element (see 7.36.10).
•	•	•	members-list-element	Single enumeration list elements of MEMBERS may be changed.
		•	LABEL	Lexical element (see 7.36.9).
	•	•	HELP	Lexical element (see 7.36.8).
	•	•	RESPONSE_CODES	Lexical element (see 7.36.13).
	•	•	VALIDITY	Lexical element (see 7.36.16).

Lexical structure — REFERENCE_ARRAY

REFERENCE_ARRAY identifier, [[ADD, DELETE, REDEFINE], [ELEMENTS, elements-list-element, HELP, LABEL, VALIDITY]]+

The redefinition rules of REFERENCE_ARRAY are specified in Table 98 and Table 122.

Table 122 – Redefinition rules for REFERENCE_ARRAY attributes

A	D	R	Attribute	Description
		•	ELEMENTS	Lexical element (see 7.27.2).

A	D	R	Attribute	Description
•	•	•	elements-list-element	Single enumeration list elements of ELEMENTS may be changed.
	•	•	LABEL	Lexical element (see 7.36.9).
	•	•	HELP	Lexical element (see 7.36.8).
	•	•	VALIDITY	Lexical element (see 7.36.16).

Lexical structure — RESPONSE_CODES

RESPONSE_CODES identifier, [[ADD, DELETE, REDEFINE], response-code-list-element]+

The redefinition rules of RESPONSE_CODES are specified in Table 98 and Table 123.

Table 123 – Redefinition rules for RESPONSE_CODES attributes

A	D	R	Attribute	Description
•	•	•	response-code-list-element	Single enumeration list elements of RESPONSE_CODE may be changed.

Lexical structure — SOURCE

SOURCE identifier, [[ADD, DELETE, REDEFINE], [MEMBERS, members-list-element, EMPHASIS, EXIT_ACTIONS, HELP, INIT_ACTIONS, LABEL, LINE_COLOR, LINE_TYPE, REFRESH_ACTIONS, VALIDITY, Y_AXIS]]+

The redefinition rules of SOURCE are specified in Table 98 and Table 124.

Table 124 – Redefinition rules for SOURCE attributes

A	D	R	Attribute	Description
		•	MEMBERS	Lexical element (see 7.36.12).
•	•	•	members-list-element	Single enumeration list elements of MEMBERS may be changed.
	•	•	EMPHASIS	Lexical element (see 7.36.5).
	•	•	EXIT_ACTIONS	Lexical element (see 7.30.2.1).
	•	•	HELP	Lexical element (see 7.36.8).
	•	•	INIT_ACTIONS	Lexical element (see 7.30.2.2).
	•	•	LABEL	Lexical element (see 7.36.9).
	•	•	LINE_COLOR	Lexical element (see 7.36.10).
	•	•	LINE_TYPE	Lexical element (see 7.36.11).
	•	•	REFRESH_ACTIONS	Lexical element (see 7.30.2.3).
	•	•	VALIDITY	Lexical element (see 7.36.16).
	•	•	Y_AXIS	Lexical element (see 7.30.2.4).

Lexical structure — TEMPLATE

TEMPLATE identifier, [[DELETE, REDEFINE], [DEFAULT_VALUES, LABEL, HELP]]+

The redefinition rules of TEMPLATE are specified in Table 98 and Table 125.

Table 125 – Redefinition rules for TEMPLATE attributes

A	D	R	Attribute	Description
		•	DEFAULT_VALUES	Lexical element (see 7.31.2).
		•	LABEL	Lexical element (see 7.36.9).
	•	•	HELP	Lexical element (see 7.36.8).

Lexical structure — VALUE_ARRAY

VALUE_ARRAY identifier, [[DELETE, REDEFINE], [LABEL, NUMBER_OF_ELEMENTS, TYPE, HELP, RESPONSE_CODES, VALIDITY]]+

The redefinition rules of VALUE_ARRAY are specified in Table 98 and Table 126.

Table 126 – Redefinition rules for VALUE_ARRAY attributes

A	D	R	Attribute	Description
		•	LABEL	Lexical element (see 7.36.9).
		•	NUMBER_OF_ELEMENTS	Lexical element (see 7.32.2.1).
		•	TYPE	Lexical element (see 7.32.2.2).
	•	•	HELP	Lexical element (see 7.36.8).
	•	•	RESPONSE_CODES	Lexical element (see 7.36.13).
	•	•	VALIDITY	Lexical element (see 7.36.16).

Lexical structure — VARIABLE

VARIABLE identifier, [[ADD, DELETE, REDEFINE], [CLASS, LABEL, TYPE, DEFAULT_VALUE, DISPLAY_FORMAT, EDIT_FORMAT, INITIAL_VALUE, MAX_VALUE, MIN_VALUE, SCALING_FACTOR, TIME_FORMAT, TIME_SCALE, type-list-element, CONSTANT_UNIT, HANDLING, HELP, POST_EDIT_ACTIONS, POST_READ_ACTIONS, POST_WRITE_ACTIONS, PRE_EDIT_ACTIONS, PRE_READ_ACTIONS, PRE_WRITE_ACTIONS, READ_TIMEOUT, REFRESH_ACTIONS, RESPONSE_CODES, STYLE, VALIDITY, WRITE_TIMEOUT]]+

The redefinition rules are specified in Table 98 and Table 127.

Table 127 – Redefinition rules for VARIABLE attributes

A	D	R	Attribute	Description
		•	CLASS	Lexical element (see 7.33.2.1).
	•	•	LABEL	Lexical element (see 7.36.9).
			TYPE	Lexical element (see 7.33.2.2).
	•	•	DEFAULT_VALUE	Lexical element (see 7.33.2.2).
	•	•	DISPLAY_FORMAT	Lexical element (see 7.33.2.2).
	•	•	EDIT_FORMAT	Lexical element (see 7.33.2.2).
	•	•	INITIAL_VALUE	Lexical element (see 7.33.2.2).
	•	•	MAX_VALUE	Lexical element (see 7.33.2.2).
	•	•	MIN_VALUE	Lexical element (see 7.33.2.2).
	•	•	SCALING_FACTOR	Lexical element (see 7.33.2.2).
	•	•	TIME_FORMAT	Lexical element (see 7.33.2.2).
	•	•	TIME_SCALE	Lexical element (see 7.33.2.2).
•	•	•	type-list-element	Single enumeration list elements of every TYPE may be changed.
	•	•	CONSTANT_UNIT	Lexical element (see 7.33.2.3).
	•	•	HANDLING	Lexical element (see 7.36.6).
	•	•	HELP	Lexical element (see 7.36.8).
	•	•	POST_EDIT_ACTIONS	Lexical element (see 7.33.2.6).
	•	•	POST_READ_ACTIONS	Lexical element (see 7.33.2.7).
	•	•	POST_WRITE_ACTIONS	Lexical element (see 7.33.2.8).
	•	•	PRE_EDIT_ACTIONS	Lexical element (see 7.33.2.9).
	•	•	PRE_READ_ACTIONS	Lexical element (see 7.33.2.10).

A	D	R	Attribute	Description
	•	•	PRE_WRITE_ACTIONS	Lexical element (see 7.33.2.11).
	•	•	READ_TIMEOUT	Lexical element (see 7.33.2.12).
	•	•	REFRESH_ACTIONS	Lexical element (see 7.33.2.13).
	•	•	RESPONSE_CODES	Lexical element (see 7.36.13).
	•	•	STYLE	Lexical element (see 7.33.2.13).
	•	•	VALIDITY	Lexical element (see 7.36.16).
	•	•	WRITE_TIMEOUT	Lexical element (see 7.33.2.12).

Lexical structure — VARIABLE_LIST

VARIABLE_LIST identifier, [[ADD, DELETE, REDEFINE], [MEMBERS, members-list-element, HELP, LABEL, RESPONSE_CODES]]+

The redefinition rules of VARIABLE_LIST are specified in Table 98 and Table 128.

Table 128 – Redefinition rules for VARIABLE_LIST attributes

A	D	R	Attribute	Description
		•	MEMBERS	Lexical element (see 7.36.12).
•	•	•	members-list-element	Single enumeration list elements of MEMBERS may be changed.
	•	•	HELP	Lexical element (see 7.36.8).
	•	•	LABEL	Lexical element (see 7.36.9).
	•	•	RESPONSE_CODES	Lexical element (see 7.36.13).

Lexical structure — WAVEFORM

WAVEFORM identifier, [[ADD, DELETE, REDEFINE], [TYPE, X_VALUES, Y_VALUES, NUMBER_OF_POINTS, X_INITIAL, X_INCREMENT, EMPHASIS, HANDLING, HELP, INIT_ACTIONS, KEY_POINTS, LABEL, LINE_COLOR, LINE_TYPE, REFRESH_ACTIONS, VALIDITY, Y_AXIS]]+

The redefinition rules of WAVEFORM are specified in Table 98 and Table 129.

Table 129 – Redefinition rules for WAVEFORM attributes

A	D	R	Attribute	Description
		•	TYPE	Lexical element (see 7.35.2.1).
•		•	X_VALUES	Lexical element (see 7.35.2.1).
•		•	Y_VALUES	Lexical element (see 7.35.2.1).
	•	•	NUMBER_OF_POINTS	Lexical element (see 7.35.2.1).
		•	X_INITIAL	Lexical element (see 7.35.2.1).
		•	X_INCREMENT	Lexical element (see 7.35.2.1).
	•	•	EMPHASIS	Lexical element (see 7.36.5).
	•	•	EXIT_ACTIONS	Lexical element (see 7.35.2.2).
	•	•	HANDLING	Lexical element (see 7.36.6).
	•	•	HELP	Lexical element (see 7.36.8).
	•	•	INIT_ACTIONS	Lexical element (see 7.35.2.3).
	•	•	KEY_POINTS	Lexical element (see 7.35.2.4).
	•	•	LABEL	Lexical element (see 7.36.9).
	•	•	LINE_COLOR	Lexical element (see 7.36.10).
	•	•	LINE_TYPE	Lexical element (see 7.36.11).

A	D	R	Attribute	Description
	•	•	REFRESH_ACTIONS	Lexical element (see 7.35.2.5).
	•	•	VALIDITY	Lexical element (see 7.36.16).
	•	•	Y_AXIS	Lexical element (see 7.35.2.6).

7.20 INTERFACE

7.20.1 General Structure

Purpose

The INTERFACE specifies a set of basic constructs of components that are accessible by other components. An interface can contain declarations of basic constructs e.g. VARIABLES, MENUs.

All supplied INTERFACES of a component should be listed in the SUPPLIED_INTERFACE attribute of the COMPONENT.

If INTERFACES are defined for a COMPONENT, then access is restricted. Other components may only access EDD basic constructs (e.g. VARIABLES) defined in an interface.

If no interface is defined for the component, then access is allowed to all basic constructs.

Lexical structure

INTERFACE identifier [LABEL, HELP, DECLARATION]

The attributes of INTERFACE are specified in Table 130.

Table 130 – INTERFACE attribute

Usage	Attribute	Description
m	DECLARATION	Lexical element (see 7.20.2).
m	LABEL	Lexical element (see 7.36.9).
o	HELP	Lexical element (see 7.36.8).

7.20.2 Specific attribute – DECLARATION

Purpose

The DECLARATION attribute specifies the basic constructs included in the interface.

Lexical structure

DECLARATION [IMAGE, MENU, METHOD, VARIABLE] +

The attributes of DECLARATION are specified in Table 131.

Table 131 – DECLARATION attribute

Usage	Attribute	Description
o	IMAGE	IMAGE declarations (see 7.18).
o	MENU	MENU declarations (see 7.23).
o	METHOD	METHOD declarations (see 7.24).
o	VARIABLE	VARIABLE declarations (see 7.33).

7.21 LIKE

Purpose

LIKE is used to create a new EDD instance based on an existing item. LIKE makes a copy of the existing item type with all attributes.

Lexical structure

LIKE identifier, reference, attribute-redefinition+, item-type

The attributes are specified in Table 132.

Table 132 – LIKE attributes

Usage	Attribute	Description
m	identifier	The name of the new EDD instance.
m	reference	A reference to an EDD instance which shall be copied.
o	attribute-redefinition	Specifies whether an attribute is added, deleted or redefined (see 7.19.2). Only existing attributes can be deleted or redefined.
o	item-type	One of the elements listed in Table 47. Even if item-type is present it is not interpreted.

7.22 LIST

7.22.1 General structure

Purpose

LIST describes a sequence of elements where each element of the sequence is defined by a EDDL construct instance.

Lexical structure

LIST identifier [TYPE, CAPACITY, COUNT, HELP, LABEL, VALIDITY]

The attributes of LIST are specified in Table 133.

Table 133 – LIST attributes

Usage	Attribute	Description
m	TYPE	Lexical element (see 7.22.2.1).
o	CAPACITY	Lexical element (see 7.22.2.2).
o	COUNT	Lexical element (see 7.22.2.3).
o	HELP	Lexical element (see 7.36.8).
o	LABEL	Lexical element (see 7.36.9).
o	VALIDITY	Lexical element (see 7.36.16).

7.22.2 Specific attributes

7.22.2.1 TYPE

Purpose

The TYPE attribute is a reference to a EDDL construct instance (for example, VARIABLE, COLLECTION). All attributes from the EDDL construct apply to each of the LIST elements.

Lexical structure

TYPE reference

The attribute of TYPE is specified in Table 134.

Table 134 – TYPE attribute

Usage	Attribute	Description
m	reference	A reference to a VARIABLE instance, a RECORD instance, a VALUE_ARRAY instance, a COLLECTION instance or a REFERENCE_ARRAY instance.

7.22.2.2 CAPACITY

Purpose

The CAPACITY attribute specifies the maximum number of elements that may be stored in the list. When a LIST is part of a FILE, the CAPACITY shall not be specified since the capacity is only limited by the resources of the EDD application.

Lexical structure

CAPACITY `unsigned_integer`

The attribute of CAPACITY is specified in Table 135.

Table 135 – CAPACITY, COUNT attribute

Usage	Attribute	Description
m	<code>unsigned_integer</code>	An unsigned integer constant.

7.22.2.3 COUNT

Purpose

The COUNT attribute specifies the numbers of elements currently stored in the list. When a LIST is part of a FILE, the COUNT shall not be specified since only the EDD application knows how many elements are currently in the list.

If COUNT is specified, it defines the size that it has at creation time of the LIST instance. If it is not defined, a LIST instance with zero elements is created.

Lexical structure

COUNT `(unsigned_integer)<exp>`

The attribute of COUNT is specified in Table 135.

7.23 MENU

7.23.1 General structure

Purpose

MENU organizes variables, methods, and other items specified in the EDDL into a hierarchical structure. A user application may use the MENU ITEMS to display and enter information in an organized and consistent fashion.

Lexical structure

MENU `identifier [ITEMS, LABEL, ACCESS, HELP, POST_EDIT_ACTIONS, POST_READ_ACTIONS, POST_WRITE_ACTIONS, PRE_EDIT_ACTIONS, PRE_READ_ACTIONS, PRE_WRITE_ACTIONS, PURPOSE, ROLE, STYLE, VALIDITY]`

The attributes of MENU are specified in Table 136.

Table 136 – MENU attribute

Usage	Attribute	Description
m	ITEMS	Lexical element (see 7.23.2.1).
m	LABEL	Lexical element (see 7.36.9).
o	ACCESS	Lexical element (see 7.23.2.2).
o	HELP	Lexical element (see 7.36.8).
o	POST_EDIT_ACTIONS	Lexical element (see 7.23.2.3).
o	POST_READ_ACTIONS	Lexical element (see 7.23.2.4).
o	POST_WRITE_ACTIONS	Lexical element (see 7.23.2.5).
o	PRE_EDIT_ACTIONS	Lexical element (see 7.23.2.6).

Usage	Attribute	Description
o	PRE_READ_ACTIONS	Lexical element (see 7.23.2.7).
o	PRE_WRITE_ACTIONS	Lexical element (see 7.23.2.8).
c	PURPOSE	Lexical element (see 7.23.2.9).
o	ROLE	Lexical element (see 7.23.2.10).
o	STYLE	Lexical element (see 7.23.2.11).
o	VALIDITY	Lexical element (see 7.36.16).

7.23.2 Specific attributes

7.23.2.1 ITEMS

Purpose

The ITEMS attribute specifies selected elements of the EDD which shall be displayed to the user plus optional qualifiers for each element. These qualifiers (DISPLAY_VALUE, HIDDEN, READ_ONLY, NO_LABEL, NO_UNIT, REVIEW) are used to control the item processing.

NOTE 1 The MENU items are presented to the user in the order they appear.

NOTE 2 For vertical menus, the first item is displayed on top and the last item on the bottom. For horizontal menus, the first item is displayed on the left and the last item on the right.

COLUMNBREAK and ROWBREAK shall not appear within a conditional.

Lexical structure

ITEMS [(reference, DISPLAY_VALUE, HIDDEN, READ_ONLY, NO_LABEL, NO_UNIT, REVIEW)<exp>, SEPARATOR, ROWBREAK, COLUMNBREAK, string]+

The attributes of ITEMS are specified in Table 137.

Table 137 – ITEMS attribute

Usage	Attribute	Description
m	reference	A reference to a VARIABLE, a BLOCK_A PARAMETERS, an element of BLOCK_A PARAMETERS, a CHART, a COLLECTION, an EDIT_DISPLAY, a GRAPH, a GRID, an IMAGE, a LIST, a MENU, a METHOD, a METHOD with arguments, a RECORD, a REFERENCE_ARRAY, or a VALUE_ARRAY.
c/o	DISPLAY_VALUE	Only used with VARIABLE; o used in all other cases. Specifies that the value of the corresponding VARIABLE is displayed in addition to its LABEL.
o	HIDDEN	The item is not displayed.
c/o	READ_ONLY	Only used with VARIABLE; o used in all other cases. Specifies that the value shall be displayed but it may not be modified (regardless of its HANDLING attribute).
c/o	NO_LABEL	Only used with VARIABLE; o used in all other cases. Specifies that the label of the VARIABLE is not be displayed; DISPLAY_VALUE is implied.
c/o	NO_UNIT	Only used with VARIABLE; o used in all other cases. Specifies that the engineering unit of the VARIABLE is not be displayed; DISPLAY_VALUE is implied.
c/o	REVIEW	Only used with MENU; o used in all other cases. REVIEW is used to present a large quantity of data e.g. for review purpose.
c/o	INLINE	Only used with IMAGE; o used in all other cases. If the IMAGE is in a MENU with a STYLE attribute equal to WINDOW, DIALOG, PAGE or GROUP and the INLINE qualifier is not specified, the IMAGE should span with width of the MENU; otherwise, the IMAGE should be displayed inline with other ITEMS of the MENU.
o	SEPARATOR	The menu items before and after the SEPARATOR should be separated, e.g., via a horizontal line.
c/o	COLUMNBREAK	Only used within a MENU whose STYLE attribute is WINDOW, DIALOG, PAGE, or GROUP; specifies a new column should be started.

Usage	Attribute	Description
c/o	ROWBREAK	Only used within a MENU whose STYLE attribute is WINDOW, DIALOG, PAGE, or GROUP; specifies a new row should be started.
o	string	Static text that should be displayed on the menu.

7.23.2.2 ACCESS

Purpose

The ACCESS attribute specifies whether the items displayed or used in the MENU are read from the device (ONLINE) or locally (OFFLINE). If no ACCESS attribute is defined, ONLINE is the default.

Lexical structure

ACCESS [(OFFLINE, ONLINE) <exp>]

The attributes of ACCESS are specified in Table 138.

Table 138 – ACCESS attribute

Usage	Attribute	Description
s	OFFLINE	The values of the items displayed in the MENU are read locally.
s	ONLINE	The values of the items displayed in the MENU should be read from the device and refreshed with an appropriate interval.

NOTE The implementation is responsible for deciding when the transfer of modified values to a device should take place (typically after a single entry or MENU completion for an ONLINE ACCESS attribute, or via a separate menu to update the device).

7.23.2.3 POST_EDIT_ACTIONS

Purpose

The POST_EDIT_ACTIONS attribute specifies METHODS that shall be executed after the user has finished processing the MENU. If the MENU is aborted, POST_EDIT_ACTIONS are not executed. The specified METHODS shall be executed in the order they appear. If a METHOD exits for an unplanned reason, the following METHODS are not executed.

Any POST_EDIT_ACTIONS associated with a referenced entity in the ITEMS shall be executed whenever a new value has been entered for the entity. The POST_EDIT_ACTIONS of the MENU are executed only after all entity specific POST_EDIT_ACTIONS have been completed.

The POST_EDIT_ACTIONS attribute shall not be expressed using a conditional IF, IF-ELSE, SELECT. If conditional behavior is required, the methods invoked by the action shall accommodate it.

Lexical structure

POST_EDIT_ACTIONS [reference]+, DEFINITION

The attributes of POST_EDIT_ACTIONS are specified in Table 139.

Table 139 – POST_EDIT_ACTIONS, PRE_EDIT_ACTIONS, POST_READ_ACTIONS, PRE_READ_ACTIONS, POST_WRITE_ACTIONS, PRE_WRITE_ACTIONS attributes

Usage	Attribute	Description
o	reference	A reference to a METHOD instance or a METHOD instance with arguments.
o	DEFINITION	Lexical element (see 7.36.4).

7.23.2.4 POST_READ_ACTIONS

Purpose

The POST_READ_ACTIONS attribute specifies METHODS that shall be executed after all referenced entities in the ITEMS attribute have been read and processed. The specified METHODS shall be executed in the order they appear. If a METHOD exits for an unplanned reason, the following METHODS are not executed and MENU activation is cancelled.

Any POST_READ_ACTIONS associated with a referenced entity in the ITEMS shall be executed whenever a new value has been read for the entity. The POST_READ_ACTIONS of the MENU shall be executed after all entity specific POST_READ_ACTIONS have been completed.

The POST_READ_ACTIONS attribute shall not be expressed using a conditional (IF, IF-ELSE, SELECT). If conditional behavior is required, the methods invoked by the action shall accommodate it.

Lexical structure

POST_READ_ACTIONS [reference]+, DEFINITION

The attributes of POST_READ_ACTIONS are specified in Table 139.

7.23.2.5 POST_WRITE_ACTIONS

Purpose

The POST_WRITE_ACTIONS attribute specifies METHODS that shall be executed after all referenced entities in the ITEMS attribute have been written and processed. The specified METHODS shall be executed in the order they appear. If a METHOD exits for an unplanned reason, the following METHODS are not executed.

Any POST_WRITE_ACTIONS associated with a referenced entity in the ITEMS shall be executed whenever a new value has been written to the entity. The POST_WRITE_ACTIONS of the MENU shall be executed after all entity specific POST_WRITE_ACTIONS have been completed.

The POST_WRITE_ACTIONS attribute shall not be expressed using a conditional (IF, IF-ELSE, SELECT). If conditional behavior is required, the methods invoked by the action shall accommodate it.

Lexical structure

POST_WRITE_ACTIONS [reference]+, DEFINITION

The attributes of POST_WRITE_ACTIONS are specified in Table 139.

7.23.2.6 PRE_EDIT_ACTIONS

Purpose

The PRE_EDIT_ACTIONS attribute specifies METHODS that shall be executed immediately the MENU is activated. The specified METHODS shall be executed in the order they appear. If a METHOD exits for an unplanned reason, the following METHODS are not executed and MENU activation is cancelled.

The PRE_EDIT_ACTIONS of the MENU shall be executed before any of the defined PRE_EDIT_ACTIONS of the referenced entities. After the MENU PRE_EDIT_ACTIONS are completed, any PRE_EDIT_ACTIONS for the referenced entities shall be executed.

The PRE_EDIT_ACTIONS attribute shall not be expressed using a conditional (IF, IF-ELSE, SELECT). If conditional behavior is required, the methods invoked by the action shall accommodate it.

Lexical structure

PRE_EDIT_ACTIONS [reference]+, DEFINITION

The attributes of PRE_EDIT_ACTIONS are specified in Table 139.

7.23.2.7 PRE_READ_ACTIONS

Purpose

The PRE_READ_ACTIONS attribute specifies METHODS that shall be executed before any referenced entities in the ITEMS attribute are read or processed. The specified METHODS shall be executed in the order they appear. If a METHOD exits for an unplanned reason, the following METHODS are not executed and MENU activation is cancelled. After the MENU PRE_READ_ACTIONS are completed, any PRE_READ_ACTIONS for the referenced entities shall be executed.

The PRE_READ_ACTIONS attribute shall not be expressed using a conditional (IF, IF-ELSE, SELECT). If conditional behavior is required, the methods invoked by the action shall accommodate it.

Lexical structure

PRE_READ_ACTIONS [reference] +, DEFINITION

The attributes of PRE_READ_ACTIONS are specified in Table 139.

7.23.2.8 PRE_WRITE_ACTIONS

Purpose

The PRE_WRITE_ACTIONS attribute specifies METHODS that shall be executed before any referenced entities in the ITEMS attribute are written or processed. The specified METHODS shall be executed in the order they appear. If a METHOD exits for an unplanned reason, the following METHODS are not executed. After the MENU PRE_WRITE_ACTIONS are completed, the PRE_WRITE_ACTIONS for the referenced entities shall be executed.

The PRE_WRITE_ACTIONS attribute shall not be expressed using a conditional (IF, IF-ELSE, SELECT). If conditional behavior is required, the methods invoked by the action shall accommodate it.

Lexical structure

PRE_WRITE_ACTIONS [reference] +, DEFINITION

The attributes of PRE_WRITE_ACTIONS are specified in Table 139.

7.23.2.9 PURPOSE

Purpose

The PURPOSE attribute defines the application purpose of the MENU. This indicates how the MENU is used, for example, for a tabular presentation, diagnostic actions, and upload/download list.

Lexical structure

PURPOSE [(CATALOG, DIAGNOSE, LOAD_TO_APPLICATION, LOAD_TO_DEVICE, MENU, PROCESS_VALUE, SIMULATION, string, TABLE)<exp>] +

The attributes of PURPOSE are specified in Table 140.

Table 140 – PURPOSE attribute

Usage	Attribute	Description
s	CATALOG	The elements of the ITEMS attribute of a MENU instance are references to catalog specific information (for example, an EDD variable containing the ordering number).
s	DIAGNOSE	The MENU is used for diagnostic actions.
s	LOAD_TO_APPLICATION	The MENU is used for listing VARIABLES whose values are sent from the device to an application.
s	LOAD_TO_DEVICE	The MENU is used for listing VARIABLES whose values are sent to the device.
s	MENU	A MENU which contains other MENU items or control items normally

Usage	Attribute	Description
		presented using horizontal rows.
s	PROCESS_VALUE	The MENU is used to display process values.
s	SIMULATION	The MENU is used for simulations actions.
s	string	A user-defined PURPOSE for the MENU.
s	TABLE	A MENU which contains other MENU items or VARIABLEs normally presented using vertical columns.

NOTE The PURPOSE attribute MENU should be used for a horizontal data and function representation (for example, function keys, application menus). TABLE should be used for a vertical data and function representation (for example, tables).

7.23.2.10 ROLE

Purpose

The ROLE attribute specifies in which role context the MENU is used. If a MENU contains sub-menus, only these submenus are displayed which have the same ROLE. If a MENU has no ROLE attribute, the MENU shall be displayed wherever it is referenced.

Lexical structure

ROLE [(string)<exp>]+

The attribute of ROLE is specified in Table 141.

Table 141 – ROLE attribute

Usage	Attribute	Description
m	string	A string which indicates the ROLE of the MENU.

NOTE An EDD may contain one or more of the following ROLES: MAINTENANCE, SPECIALIST, DIAGNOSE, HANDHELD, SERVICE, MANUFACTURER, OBSERVER.

7.23.2.11 STYLE

Purpose

The STYLE attribute specifies how the MENU is displayed.

When the STYLE attribute is DIALOG, WINDOW, PAGE or GROUP, the DISPLAY_VALUE qualifier (see 7.23.2.1) is implied on each item on the MENU.

If one or more MENUs are active and a DIALOG MENU is activated, the DIALOG MENU takes precedence and shall be closed before any other MENU can be used.

Lexical structure

STYLE (DIALOG, WINDOW, PAGE, GROUP, MENU, TABLE, string)

The attributes of STYLE are specified in Table 142.

Table 142 – STYLE attribute

Usage	Attribute	Description
s	DIALOG	No other MENU can be activated while this MENU, or any of its submenus, are active. The MENU shall be closed before interactions with other MENUs are possible, i.e. specifies that the items of the MENU should be displayed as a modal dialog box.
s	WINDOW	Other MENUs can be activated while this MENU is active. Interactions with all activated MENUs having WINDOW STYLE attributes are possible, i.e. specifies that the items of the MENU should be displayed as a modeless window.
s	PAGE	The items of the MENU should be displayed as a tabbed property page.
s	GROUP	The items of the MENU should be displayed as a group box.

Usage	Attribute	Description
s	MENU	The items of the MENU should be displayed as either a pulldown or popup menu.
s	TABLE	The items of the MENU should be displayed in vertical columns.
s	string	A tool specific MENU style. If a tool cannot interpret the string, this STYLE attribute shall be ignored.

7.23.3 Sequence diagrams for actions

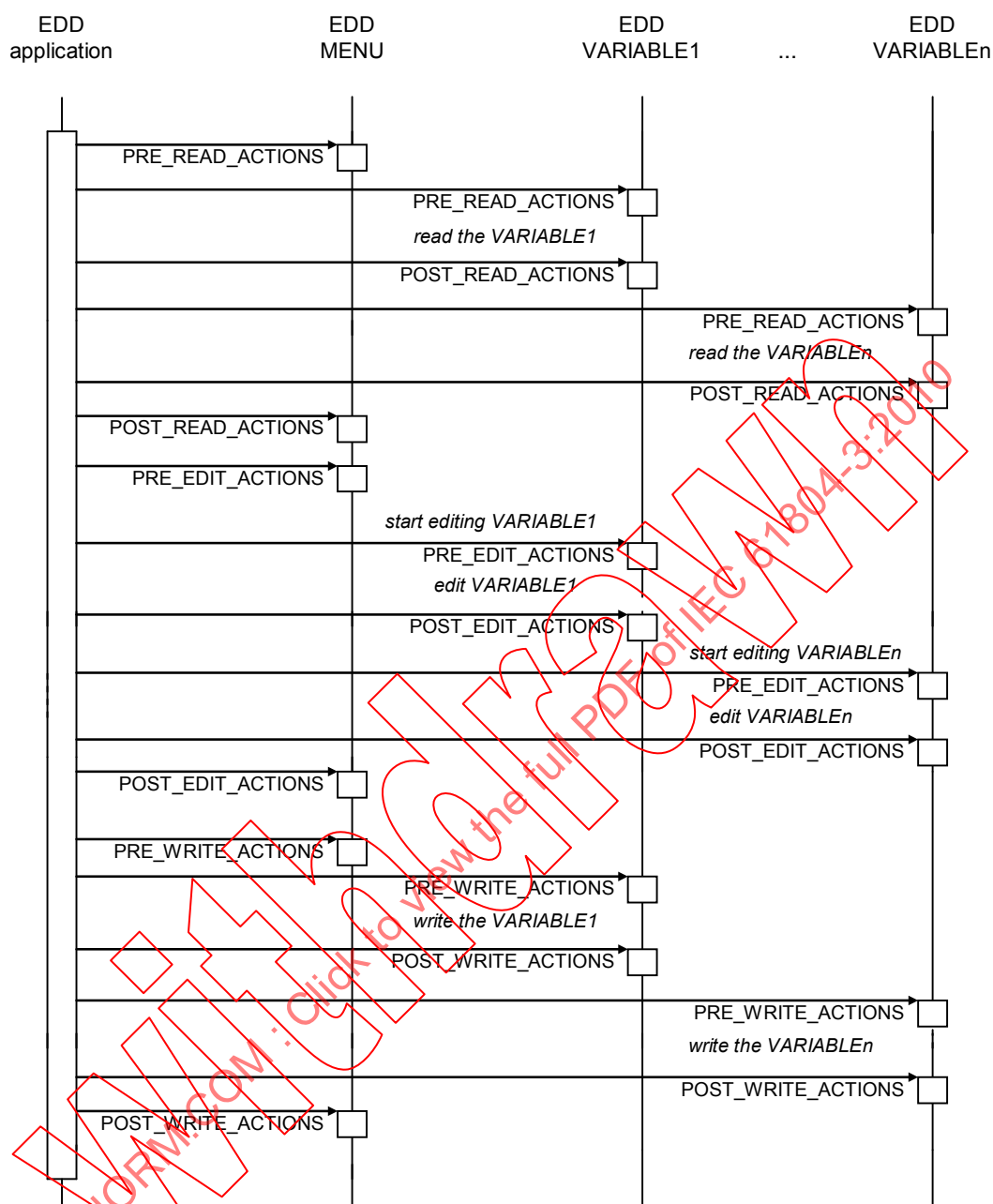
Figure 28 show the sequence diagrams for actions related to MENUs and their contained VARIABLES and METHODS. Actions are executed and completed in sequence moving down a diagram. If any action fails or aborts for an unplanned reason, the remaining actions are not executed and the MENU is cancelled.

When the MENU has the ACCESS OFFLINE (data coming from a local data base), the pre/post read/write actions for the VARIABLE are not executed.

When the MENU has the ACCESS ONLINE (data is read from or written to the device), the pre/post read/write actions for the VARIABLE shall be executed in the order show in the sequence.

The pre/post edit actions shall always be executed in case of ACCESS OFFLINE and ONLINE.

In the case of online access, VARIABLE values may be changed and immediately written unless they are in a WRITE_AS_ONE relation. Any WRITE_AS_ONE entities are written together. After the WRITE_AS_ONE VARIABLES have been written, the POST_WRITE_ACTIONS are executed.



IEC 2409/10

Figure 28 – MENU activation

7.24 METHOD

7.24.1 General structure

Purpose

A METHOD is used to define a subroutine which is executed in the EDD application.

NOTE A METHOD is used to specify consistency checks, conversion algorithms between EDD variables, user acknowledges or specific device interactions.

Lexical structure

METHOD identifier [(parameter-type parameter-name)+] [DEFINITION, ACCESS, CLASS, HELP, LABEL, TYPE, VALIDITY]

The attributes of METHOD are specified in Table 143.

Table 143 – METHOD attributes

Usage	Attribute	Description
m	DEFINITION	Lexical element (see 7.36.4).
o	ACCESS	Lexical element (see 7.24.2.1).
o	CLASS	Lexical element (see 7.24.2.2).
o	HELP	Lexical element (see 7.36.8).
o	LABEL	Lexical element (see 7.36.9).
o	TYPE	Lexical element (see 7.24.2.3).
o	VALIDITY	Lexical element (see 7.36.16).
o	parameter-type	The type of the parameter (see Table 134).
o	parameter-name	The name of the parameter in the form of an Identifier.

7.24.2 Specific attributes**7.24.2.1 ACCESS****Purpose**

The ACCESS attribute specifies whether the VARIABLE values stored in the device or the offline data set are used within the METHOD DEFINITION. If ACCESS is not defined, the default is ONLINE.

Lexical structure

ACCESS (OFFLINE, ONLINE) <exp>

The attributes of ACCESS are specified in Table 144.

Table 144 – ACCESS attributes

Usage	Attribute	Description
s	OFFLINE	The values of the variables used within the METHOD definition are read or written from/to the offline data set.
s	ONLINE	The values of the variables used within the METHOD definition are actually read from the device. Changes of VARIABLE values are written with Builtins.

7.24.2.2 CLASS**Purpose**

The CLASS attribute specifies the effect of a METHOD on a field device. This attribute is intended to be used by the EDD application to implement permission levels and organize how METHODS are presented. METHOD classes are identical to VARIABLE classes, except that they shall not be LOCAL.

Lexical structure

CLASS [ALARM, ANALOG_OUTPUT, COMPUTATION, CONTAINED, DEVICE, DIAGNOSTIC, DISCRETE, DYNAMIC, FREQUENCY, HART, INPUT, LOCAL_DISPLAY, OUTPUT, OPERATE, SENSOR_CORRECTION, TUNE, SERVICE] +

The attributes of CLASS are specified in Table 163.

7.24.2.3 TYPE**Purpose**

The TYPE attribute specifies the type of the returned value.

Lexical structure

TYPE [char, unsigned char, int, unsigned int, long, unsigned long, float, double, time_t, DD_STRING]

The attributes of TYPE are specified in Table 145.

Table 145 – TYPE attributes

Usage	Attribute	Description
s	char	The METHOD returns a value of type char.
s	unsigned char	The METHOD returns a value of type unsigned char.
s	short	The METHOD returns a value of type short.
s	unsigned short	The METHOD returns a value of type unsigned short.
s	int	The METHOD returns a value of type int.
s	unsigned int	The METHOD returns a value of type unsigned int.
s	long	The METHOD returns a value of type long.
s	unsigned long	The METHOD returns a value of type unsigned long.
s	float	The METHOD returns a value of type float.
s	double	The METHOD returns a value of type double.
s	time_t	The METHOD returns a value of type time_t.
s	DD_STRING	The METHOD returns a value of type DD_STRING.

7.25 PROGRAM

7.25.1 General structure

Purpose

PROGRAM allows remote control for the execution of a program.

EXAMPLE A device could download a PROGRAM into a domain of another device using a download service and then remotely operate the PROGRAM by issuing a service request. PROGRAMs should be created, deleted, started, stopped, etc.

Lexical structure

PROGRAM identifier [ARGUMENTS, RESPONSE_CODES]

The attributes of PROGRAM are specified in Table 146.

Table 146 – PROGRAM attributes

Usage	Attribute	Description
o	ARGUMENTS	Lexical element (see 7.25.2).
o	RESPONSE_CODES	Lexical element (see 7.36.13).

7.25.2 Specific attribute – ARGUMENTS

Purpose

The ARGUMENTS attribute can be sent to the program during start and resume operations. The PROGRAM arguments are described by the ARGUMENTS attribute. An octet string containing the values of all arguments will be sent to the program invocation object when it is started or resumed by the EDD application.

Lexical structure

ARGUMENTS [(data-item)<exp>]+

The attribute of ARGUMENT is specified in Table 147.

Table 147 – ARGUMENTS attribute

Usage	Attribute	Description
m	data-item	Either an unsigned integer constant or a VARIABLE. If a data item is an unsigned integer constant, the value of the constant appears at that position in the data field. Constant data items occupy two octets of the data field and therefore shall be in the range 0 through 65535 inclusive. If a data item is a VARIABLE, the value of the variable appears at that position in the data field. If the data field of the PROGRAM service is empty, the arguments can be omitted.

7.26 RECORD

Purpose

A RECORD is a logical group of VARIABLES. Each item in the RECORD is a reference to a VARIABLE and may be of different data types.

NOTE A RECORD may be transferred as one communication object.

Lexical structure

RECORD identifier [MEMBERS, LABEL, HELP, RESPONSE_CODES, VALIDITY]

The attributes of RECORD are specified in Table 148.

Table 148 – RECORD attributes

Usage	Attribute	Description
m	MEMBERS	Lexical element (see 7.36.10).
m	LABEL	Lexical element (see 7.36.9).
o	HELP	Lexical element (see 7.36.8).
o	RESPONSE_CODES	Lexical element (see 7.36.13).
o	VALIDITY	Lexical element (see 7.36.16).

7.27 REFERENCE_ARRAY

7.27.1 General structure

Purpose

A REFERENCE_ARRAY is a set of items of the specified item type (for example, VARIABLE or MENU).

NOTE REFERENCE_ARRAYS are not related to communication arrays (item type "VALUE_ARRAY"). Communication arrays are arrays of values.

Lexical structure

REFERENCE_ARRAY identifier [item-type, ELEMENTS, HELP, LABEL, VALIDITY]

The attributes of REFERENCE_ARRAY are specified in Table 149.

Table 149 – REFERENCE_ARRAY attribute

Usage	Attribute	Description
m	item-type	The type of items listed within the ELEMENTS attribute. It shall be one of the basic elements listed in Table 47.
m	ELEMENTS	Lexical element (see 7.27.2)
o	HELP	Lexical element (see 7.36.8)
o	LABEL	Lexical element (see 7.36.9)
o	VALIDITY	Lexical element (see 7.36.16)

7.27.2 Specific attribute – ELEMENTS

Purpose

The ELEMENTS attribute specifies a list of items. It defines for each item the item reference, its associated index, optional description and help. The description and help attributes, when used, override the corresponding specifications in the EDD item itself.

Lexical structure

ELEMENTS [integer, reference, description, help]<exp>]+

The attributes of ELEMENTS are specified in Table 150.

Table 150 – ELEMENTS attribute

Usage	Attribute	Description
m	integer	The number by which the item may be referenced. Each number shall be uniquely defined within an REFERENCE_ARRAY construct.
m	reference	A reference to a data-item instance.
o	description	A short description for the item.
o	help	Help text for the item.

7.28 Relations

7.28.1 REFRESH

Purpose

A REFRESH relation is used during online access and specifies a set of VARIABLES which should be refreshed (read from the device) whenever a VARIABLE from a specified set is modified.

NOTE A VARIABLE may have a refresh relationship with itself, implying that the VARIABLE is to be read after writing.

Lexical structure

REFRESH identifier, [(cause-parameter)<exp>]+, [(effected-parameter)<exp>]+

The attributes of REFRESH are specified in Table 151.

Table 151 – REFRESH attributes

Usage	Attribute	Description
m	cause-parameter	A list of VARIABLE instances.
m	effected-parameter	A list of VARIABLE instances, which shall be read, if one of the cause parameters has been modified. A VARIABLE of the effected parameter list may also be in the cause parameter list of another REFRESH relation.

7.28.2 UNIT

Purpose

A UNIT relation specifies a reference to a VARIABLE holding a unit code and a list of dependent VARIABLES. When the VARIABLE holding the unit code is modified, the list of effected VARIABLES/AXISs using that unit code shall be refreshed. When an effected VARIABLE/AXIS is displayed, the value of its unit code shall also be displayed.

Lexical structure

UNIT identifier, [cause-parameter]<exp>], [(effected-parameter)<exp>]+

The attributes of UNIT are specified in Table 152.

Table 152 – UNIT attributes

Usage	Attribute	Description
m	cause-parameter	A reference to a VARIABLE instance which contains the unit code.
m	effected-parameter	A list of VARIABLE/AXIS instances using that unit code.

7.28.3 WRITE_AS_ONE**Purpose**

A WRITE_AS_ONE relation specifies a list of VARIABLES that shall be modified as a group by the EDD application. This relation is used when a device requires specific VARIABLES to be examined and modified at the same time for proper operation.

NOTE This relation does not necessarily mean the VARIABLES are written to the device at the same time.

Lexical structure

WRITE_AS_ONE identifier, [(reference)<exp>]+

The attribute of WRITE_AS_ONE is specified in Table 153.

Table 153 – WRITE_AS_ONE attributes

Usage	Attribute	Description
m	reference	A list of VARIABLES, VALUE_ARRAYs and RECORDs instances that shall be modified in a group with other members of the WRITE_AS_ONE relation.

7.29 RESPONSE_CODES**Purpose**

RESPONSE_CODES specify values a device may return as error information. Each VARIABLE, RECORD, VALUE_ARRAY, VARIABLE_LIST, PROGRAM, COMMAND or DOMAIN can have its own set of RESPONSE_CODES.

Lexical structure

RESPONSE_CODES identifier, [(description, integer, [DATA_ENTRY_ERROR, DATA_ENTRY_WARNING, MISC_ERROR, MISC_WARNING, MODE_ERROR, PROCESS_ERROR, SUCCESS], help)<exp>]+

The attributes of RESPONSE_CODES are specified in Table 154.

Table 154 – RESPONSE_CODES attributes

Usage	Attribute	Description
m	description	A string that will be displayed when the RESPONSE_CODE is returned by the device.
m	integer	The returned value of the device to identify the RESPONSE_CODE.
s	DATA ENTRY ERROR	The application layer service was rejected because the data sent was invalid.
s	DATA ENTRY WARNING	The application layer service was accepted and processed with a slightly modified version of the data sent.
s	MISC ERROR	The application layer service was rejected.
s	MISC WARNING	The application layer service was accepted and processed as specified and there is additional information, unrelated to the application layer service, in which the user might be interested.
s	MODE ERROR	The application layer service was rejected because the field device was in a mode in which the application layer service could not be executed.
s	PROCESS ERROR	The application layer service was rejected because a process applied to the field device was invalid.

Usage	Attribute	Description
s	SUCCESS	The application layer service was accepted and processed as specified.
o	help	Help text for the item.

7.30 SOURCE

7.30.1 General structure

Purpose

A SOURCE describes a source of data values for a CHART.

Lexical structure

SOURCE identifier [MEMBERS, EMPHASIS, EXIT_ACTIONS, HELP, INIT_ACTIONS, LABEL, LINE_COLOR, LINE_TYPE, REFRESH_ACTIONS, VALIDITY, Y_AXIS]

The attributes of SOURCE are specified in Table 155.

Table 155 – SOURCE attributes

Usage	Attribute	Description
m	MEMBERS	Lexical element (see 7.36.12).
o	EMPHASIS	Lexical element (see 7.36.5).
o	EXIT_ACTIONS	Lexical element (see 7.30.2.1).
o	HELP	Lexical element (see 7.36.8).
o	INIT_ACTIONS	Lexical element (see 7.30.2.2).
o	LABEL	Lexical element (see 7.36.9).
o	LINE_COLOR	Lexical element (see 7.36.10).
o	LINE_TYPE	Lexical element (see 7.36.11).
o	REFRESH_ACTIONS	Lexical element (see 7.30.2.3).
o	VALIDITY	Lexical element (see 7.36.16).
o	Y_AXIS	Lexical element (see 7.30.2.4).

7.30.2 Specific attributes

7.30.2.1 EXIT_ACTIONS

Purpose

The EXIT_ACTIONS attribute specifies actions that are executed when the CHART containing the SOURCE is closed.

The specified METHODS shall be executed in the order they appear. If a METHOD exits for an unplanned reason, the remaining METHODS shall not be executed.

The EXIT_ACTIONS attribute shall not be expressed using a conditional (IF, IF-ELSE, SELECT). If conditional behavior is required, the methods invoked by the action shall accommodate it.

Lexical structure

EXIT_ACTIONS [reference]+, DEFINITION

The attributes of EXIT_ACTIONS are specified in Table 188.

7.30.2.2 INIT_ACTIONS

Purpose

The INIT_ACTIONS attribute specifies actions that are executed before the SOURCE value is initially displayed by a CHART.

The specified METHODS shall be executed in the order they appear. If a METHOD exits for an unplanned reason, the remaining METHODS shall not be executed. Furthermore, the EDD application shall not draw the SOURCE value if a METHOD exits for an unplanned reason.

The INIT_ACTIONS attribute shall not be expressed using a conditional (IF, IF-ELSE, SELECT). If conditional behavior is required, the methods invoked by the action shall accommodate it.

Lexical structure

INIT_ACTIONS [reference]+, DEFINITION

The attributes of INIT_ACTIONS are specified in Table 188.

7.30.2.3 REFRESH_ACTIONS

Purpose

The REFRESH_ACTIONS attribute specifies actions that are executed prior to the display of the value on the CHART in which the SOURCE is being displayed.

The specified METHODS shall be executed in the order they appear. If a METHOD exits for an unplanned reason, the remaining METHODS shall not be executed.

The REFRESH_ACTIONS attribute shall not be expressed using a conditional (IF, IF-ELSE, SELECT). If conditional behavior is required, the methods invoked by the action shall accommodate it.

Lexical structure

REFRESH_ACTIONS [reference]+, DEFINITION

The attributes of REFRESH_ACTIONS are specified in Table 188.

7.30.2.4 Y_AXIS

Purpose

The Y_AXIS attribute specifies the way the Y axis of the SOURCE is displayed. Each SOURCE on a CHART share a common X axis that is based on time, but each SOURCE on a CHART has its own Y_AXIS.

Lexical structure

Y_AXIS (reference)<exp>

The attribute of Y_AXIS is specified in Table 156.

Table 156 – Y_AXIS attribute

Usage	Attribute	Description
m	reference	A reference to an AXIS instance.

7.31 TEMPLATE

7.31.1 General structure

Purpose

TEMPLATE describes a set of parameter values that is used for offline configuration by the EDD application. Each TEMPLATE defines default values for application specific configuration. The DEFAULT_VALUES attribute defines the default value of parameters.

Lexical structure

TEMPLATE identifier [DEFAULT_VALUES, HELP, LABEL]

The attributes of TEMPLATE are specified in Table 157.

Table 157 – TEMPLATE attributes

Usage	Attribute	Description
m	DEFAULT_VALUES	Lexical element (see 7.31.2).
m	LABEL	Lexical element (see 7.36.9).
o	HELP	Lexical element (see 7.36.8).

7.31.2 Specific attribute – DEFAULT_VALUES

Purpose

The DEFAULT_VALUES attribute specifies the default setting for the VARIABLES. The value to be set shall be constant.

Lexical structure

DEFAULT_VALUES (reference, value)+

The attributes of DEFAULT_VALUES are specified in Table 158.

Table 158 – DEFAULT_VALUES attribute

Usage	Attribute	Description
m	reference	A reference to a VARIABLE instance, a member of a RECORD instance, an element of a VALUE_ARRAY instance.
m	value	A constant that specifies the value. The type depends on the data type specified with the reference attribute.

Referencing parameters of Block is following a Multi-Block Referencing.

7.32 VALUE_ARRAY

7.32.1 General structure

Purpose

A VALUE_ARRAY is a logical group of values. Each element of a VALUE_ARRAY shall be of the same data type and shall correspond to an EDDL construct. An element is referenced from elsewhere in the EDD via the VALUE_ARRAY identifier and the element index.

NOTE From a communication perspective, the individual elements of the VALUE_ARRAY are not treated as individual variables but simply as a grouped value.

Lexical structure

VALUE_ARRAY identifier [LABEL, NUMBER_OF_ELEMENTS, TYPE, HELP, RESPONSE_CODES, VALIDITY]

The attributes of VALUE_ARRAY are specified in Table 159.

Table 159 – VALUE_ARRAY attributes

Usage	Attribute	Description
m	LABEL	Lexical element (see 7.36.9).
m	NUMBER_OF_ELEMENTS	Lexical element (see 7.32.2.1).
m	TYPE	Lexical element (see 7.32.2.2).
o	HELP	Lexical element (see 7.36.8).
o	RESPONSE_CODES	Lexical element (see 7.36.13).
o	VALIDITY	Lexical element (see 7.36.16).

7.32.2 Specific attributes

7.32.2.1 NUMBER_OF_ELEMENTS

Purpose

The `NUMBER_OF_ELEMENTS` attribute specifies the number of elements in the `VALUE_ARRAY` as an integer greater than zero.

Lexical structure

`NUMBER_OF_ELEMENTS` (integer, expression)

The attribute of `NUMBER_OF_ELEMENTS` is specified in Table 160.

Table 160 – `NUMBER_OF_ELEMENTS` attribute

Usage	Attribute	Description
s	integer	The number of elements of a <code>VALUE_ARRAY</code> .
s	expression	An expression which shall be evaluated to a non-negative integer that is within the index range of the <code>NUMBER_OF_ELEMENTS</code> attribute (for the description of expressions, see 7.39.7).

7.32.2.2 TYPE

Purpose

The `TYPE` attribute is a reference to an EDDL construct instance (for example, `VARIABLE`, `COLLECTION`). All attributes from the EDDL construct apply to each of the `VALUE_ARRAY` elements.

Lexical structure

`TYPE` reference

The `TYPE` attribute is specified in Table 161.

Table 161 – `TYPE` attribute

Usage	Attribute	Description
m	reference	A reference to an EDD basic construct instance.

7.33 VARIABLE

7.33.1 General structure

Purpose

`VARIABLE` describes parameters contained in a device or in the EDD application.

Lexical structure

`VARIABLE` identifier, [CLASS, LABEL, TYPE, CONSTANT_UNIT, DEFAULT_VALUE, HANDLING, HELP, INITIAL_VALUE, POST_EDIT_ACTIONS, POST_READ_ACTIONS, POST_WRITE_ACTIONS, PRE_EDIT_ACTIONS, PRE_READ_ACTIONS, PRE_WRITE_ACTIONS, READ_TIMEOUT, REFRESH_ACTIONS, RESPONSE_CODES, VALIDITY, WRITE_TIMEOUT]

The attributes of `VARIABLE` are specified in Table 162.

Table 162 – `VARIABLE` attributes

Usage	Attribute	Description
m	CLASS	Lexical element (see 7.33.2.1).
o	LABEL	Lexical element (see 7.36.9).
m	TYPE	Lexical element (see 7.33.2.2).
o	CONSTANT_UNIT	Lexical element (see 7.33.2.3).

Usage	Attribute	Description
o	DEFAULT_VALUE	Lexical element (see 7.33.2.4).
o	HANDLING	Lexical element (see 7.36.6).
o	HELP	Lexical element (see 7.36.8).
o	INITIAL_VALUE	Lexical element (see 7.33.2.5).
o	POST_EDIT_ACTIONS	Lexical element (see 7.33.2.6).
o	POST_READ_ACTIONS	Lexical element (see 7.33.2.7).
o	POST_WRITE_ACTIONS	Lexical element (see 7.33.2.8).
o	PRE_EDIT_ACTIONS	Lexical element (see 7.33.2.9).
o	PRE_READ_ACTIONS	Lexical element (see 7.33.2.10).
o	PRE_WRITE_ACTIONS	Lexical element (see 7.33.2.11).
o	READ_TIMEOUT	Lexical element (see 7.33.2.12).
o	REFRESH_ACTIONS	Lexical element (see 7.33.2.13).
o	RESPONSE_CODES	Lexical element (see 7.36.13).
o	STYLE	Lexical element (see 7.33.2.14).
o	VALIDITY	Lexical element (see 7.36.15).
o	WRITE_TIMEOUT	Lexical element (see 7.33.2.12).

7.33.2 Specific attributes

7.33.2.1 CLASS

Purpose

The CLASS attribute specifies how the VARIABLE is used by the device and the EDD application for organization purposes and display.

Not all class combinations are allowed. The restrictions are defined in the profile tables (see Annex D).

NOTE The definition of CLASS is based on a model in which a device is made up to a number of different functional blocks. Each block has an input and produces an output. Blocks can be combined in different ways to produce different functionalities.

Lexical structure

CLASS [ALARM, ANALOG_INPUT, ANALOG_OUTPUT, COMPUTATION, CONTAINED, CORRECTION, DEVICE, DIAGNOSTIC, DIGITAL_INPUT, DIGITAL_OUTPUT, DISCRETE_INPUT, DISCRETE_OUTPUT, DYNAMIC, FREQUENCY_INPUT, FREQUENCY_OUTPUT, HART, INPUT, LOCAL, LOCAL_DISPLAY, OPERATE, OUTPUT, SERVICE, TEMPORARY, TUNE]+

The attributes of CLASS are specified in Table 163.

Table 163 – CLASS attributes

Usage	Attribute	Description
s	ALARM	The VARIABLE contains alarm limits.
s	ANALOG_INPUT	The VARIABLE is part of an analog input block.
s	ANALOG_OUTPUT	The VARIABLE is part of an analog output block.
s	COMPUTATION	The VARIABLE is part of a computation block.
s	CONTAINED	The VARIABLE represents the physical characteristics of the device.
s	CORRECTION	The VARIABLE is part of the correction block.
s	DEVICE	The VARIABLE represents the physical characteristics of the device.
s	DIAGNOSTIC	The VARIABLE indicates the device status.
s	DIGITAL_INPUT	The VARIABLE is part of a digital input block.

Usage	Attribute	Description
s	DIGITAL_OUTPUT	The VARIABLE is part of a digital output block.
s	DISCRETE_INPUT	The VARIABLE is part of a discrete input block.
s	DISCRETE_OUTPUT	The VARIABLE is part of a discrete output block.
s	DYNAMIC	The VARIABLE is modified by the device without stimulus from the network.
s	FREQUENCY_INPUT	The VARIABLE is part of a frequency input block.
s	FREQUENCY_OUTPUT	The VARIABLE is part of a frequency output block.
s	HART	The VARIABLE is part of the HART block which characterize the HART interface. There is only one HART block.
s	INPUT	The VARIABLE is part of an input block. An input block is a special kind of computation block which does unit conversions, scaling, and damping. The parameter of the input block parameters can be determined by the output of another block.
s	LOCAL	The VARIABLE is locally used by the EDD application. Local VARIABLES are not stored in a device, but they can be sent to a device.
s	LOCAL_DISPLAY	The VARIABLE is part of the local display block. A local display block contains the VARIABLES associated with the local interface (keyboard, display, etc.) of the device.
s	OPERATE	The VARIABLE is used to control a block's operation.
s	OPTIONAL	The VARIABLE may not be present in the device.
s	OUTPUT	The VARIABLE is part of the output block. The values of output VARIABLES may be accessed by another block input.
s	SERVICE	The VARIABLE is used when performing routine maintenance.
s	TEMPORARY	The VARIABLE is initialized to its DEFAULT_VALUE in each EDD application session. LOCAL variables may be persisted by the EDD application; TEMPORARY variables are never persisted.
s	TUNE	The VARIABLE is used to tune the algorithm of a block.

7.33.2.2 TYPE

7.33.2.2.1 General structure

Purpose

The TYPE attribute describes the format of the VARIABLES value. If TYPE is specified without DEFAULT_VALUE and INITIAL_VALUE, the column "Initial value" in Table 164 shows the initial values.

Lexical structure

TYPE [DOUBLE, FLOAT, INTEGER, UNSIGNED_INTEGER, DATE, DATE_AND_TIME, DURATION, TIME, TIME_VALUE, BIT_ENUMERATED, ENUMERATED, INDEX, OBJECT_REFERENCE, ASCII, BITSTRING, EUC, OCTET, PACKED_ASCII, PASSWORD, VISIBLE]

The attributes of TYPE are specified in Table 164.

Table 164 – TYPE attributes

Usage	Attribute	Description	Initial value
s	DOUBLE	Arithmetic type (see 7.33.2.2.2.1).	0,0
s	FLOAT	Arithmetic type (see 7.33.2.2.2.1).	0,0
s	INTEGER	Arithmetic type (see 7.33.2.2.2.1).	0
s	UNSIGNED_INTEGER	Arithmetic type (see 7.33.2.2.2.1).	0
s	DATE	Date/time type (see 7.33.2.2.2.2).	1-Jan-1900
s	DATE_AND_TIME	Date/time type (see 7.33.2.2.2.2).	1-Jan-1900 00:00:00

Usage	Attribute	Description	Initial value
s	DURATION	Date/time type (see 7.33.2.2.2.2).	0 days, 0 hours, 0 minutes, 0 seconds
s	TIME	Date/time type (see 7.33.2.2.2.2).	00:00:00
s	TIME_VALUE(8)	Date/time type (see 7.33.2.2.2.2).	1-Jan-1972 00:00:00
s	TIME_VALUE(4)	Date/time type (see 7.33.2.2.2.2).	00:00:00
s	BIT_ENUMERATED	Enumeration type (see 7.33.2.2.2.3).	0
s	ENUMERATED	Enumeration type (see 7.33.2.2.2.4).	The value of the first enumeration.
s	INDEX	Index type (see 7.33.2.2.2.5).	First element in the associated reference array.
s	OBJECT_REFERENCE	Object reference (see 7.33.2.2.2.6).	NULL
s	ASCII	String type (see 7.33.2.2.2.7).	Each character set to space.
s	BITSTRING	String type (see 7.33.2.2.2.7).	Each character set to space.
s	EUC	String type (see 7.33.2.2.2.7).	Each character set to space.
s	OCTET	String type (see 7.33.2.2.2.7).	Each character set to space.
s	PACKED_ASCII	String type (see 7.33.2.2.2.7).	Each character set to question mark (?).
s	PASSWORD	String type (see 7.33.2.2.2.7).	Each character set to space.
s	VISIBLE	String type (see 7.33.2.2.2.7).	Each character set to space.

7.33.2.2.2 Specific attributes

7.33.2.2.2.1 DOUBLE, FLOAT, INTEGER, UNSIGNED_INTEGER

Purpose

VARIABLES of TYPE FLOAT and DOUBLE are single-precision basic format and double-precision basic format floating point numbers, as defined in IEEE 754, respectively. VARIABLES of TYPE INTEGER and UNSIGNED_INTEGER are signed and unsigned integer numbers, respectively.

These types are known collectively as the arithmetic types.

Lexical structure — DOUBLE

```
DOUBLE [DEFAULT_VALUE, DISPLAY_FORMAT, EDIT_FORMAT, INITIAL_VALUE,
[MAX_VALUE, m]*, [MIN_VALUE, n]*, SCALING_FACTOR], [(description, help,
value)<exp>]*
```

Lexical structure — FLOAT

```
FLOAT [DEFAULT_VALUE, DISPLAY_FORMAT, EDIT_FORMAT, INITIAL_VALUE,
[MAX_VALUE, m]*, [MIN_VALUE, n]*, SCALING_FACTOR], [(description, help,
value)<exp>]*
```

Lexical structure — INTEGER

```
INTEGER [DEFAULT_VALUE, DISPLAY_FORMAT, EDIT_FORMAT, INITIAL_VALUE,
[MAX_VALUE, m]*, [MIN_VALUE, n]*, SCALING_FACTOR, size], [description,
help, value]*
```


Lexical structure — UNSIGNED_INTEGER

UNSIGNED_INTEGER [DEFAULT_VALUE, DISPLAY_FORMAT, EDIT_FORMAT, INITIAL_VALUE, [MAX_VALUE, m]*, [MIN_VALUE, n]*, SCALING_FACTOR, size], [(description, help, value)<exp>]*

The attributes of the arithmetic types are specified in Table 165.

Table 165 – DOUBLE, FLOAT, INTEGER, UNSIGNED_INTEGER attributes

Usage	Attribute	Description
o	DEFAULT_VALUE	Purpose The DEFAULT_VALUE attribute specifies the default setting for the VARIABLE. The DEFAULT_VALUE is available for all EDDL types. If no INITIAL_VALUE is defined, the DEFAULT_VALUE is used as an INITIAL_VALUE. The DEFAULT_VALUE can also be built by an expression (see 7.40). Lexical structure DEFAULT_VALUE (double)<exp> DEFAULT_VALUE (float)<exp> DEFAULT_VALUE (integer)<exp> DEFAULT_VALUE (unsigned_integer)<exp> DEFAULT_VALUE (expression)<exp>
o	description	A short description of the item.
o	DISPLAY_FORMAT	Purpose A DISPLAY_FORMAT attribute specifies how the value of the VARIABLE is displayed. The DISPLAY_FORMAT strings are conversion specifiers of the ANSI C (see ISO/IEC 9899) functions printf and scanf, respectively. Lexical structure DISPLAY_FORMAT (string)<exp>
o	EDIT_FORMAT	Purpose An EDIT_FORMAT attribute specifies the format for the editing. The EDIT_FORMAT strings are conversion specifiers of the ANSI C (see ISO/IEC 9899) functions printf and scanf, respectively. Lexical structure EDIT_FORMAT (string)<exp>
o	help	specifies help text for the item
o	INITIAL_VALUE	Purpose When a device is instantiated for the first time, the INITIAL_VALUE of a VARIABLE is displayed. Often the DEFAULT_VALUE is conditioned and depends on other parameter settings. In this case the INITIAL_VALUE is used to reconstitute the default settings. The INITIAL_VALUE is available for all types and shall not be conditioned. Lexical structure INITIAL_VALUE (double) INITIAL_VALUE (float) INITIAL_VALUE (integer) INITIAL_VALUE (unsigned_integer)
o	MAX_VALUE	Purpose The MAX_VALUE attribute specifies the upper bound of values to which a VARIABLE shall be set. If the VARIABLE is CLASS DYNAMIC, the device should set the value of the VARIABLE inside the range specified by its minimum and maximum values. A VARIABLE may have more than one maximum value, for instance, to specify more than one range of validity. When there are multiple maximum values, an additional integer is used to number each MAX_VALUE. Every MAX_VALUE shall have an equivalent MIN_VALUE. The range specified with a pair of MIN_VALUE and MAX_VALUE may not overlap another specified range. If a MIN_VALUE/MAX_VALUE has no equivalent MAX_VALUE/MIN_VALUE, the upper/lower bound is unlimited and need not be specified. The MAX_VALUE can also be built by an expression (see 7.40). Lexical structure MAX_VALUE (double, integer)<exp> MAX_VALUE (float, integer)<exp> MAX_VALUE (integer, integer)<exp> MAX_VALUE (unsigned_integer, integer)<exp> MAX_VALUE (expression)<exp>

Usage	Attribute	Description
c	m	An increasing integer starting with one, which identifies the MAX_VALUE if more than one is present.
o	MIN_VALUE	Purpose The MIN_VALUE attribute specifies the lower bound of values to which a VARIABLE shall be set. If the VARIABLE is CLASS DYNAMIC, the device should set the value of the VARIABLE inside the range specified by its minimum and maximum values. A VARIABLE may have more than one minimum value, for instance, to specify more than one range of validity. When there are multiple minimum values, an additional integer is used to number each MIN_VALUE. Every MIN_VALUE shall have an equivalent MAX_VALUE. The range specified with a pair of MIN_VALUE and MAX_VALUE may not overlap another specified range. If a MIN_VALUE/MAX_VALUE has no equivalent MAX_VALUE/MIN_VALUE, the upper/lower bound is unlimited and need not be specified. The MIN_VALUE can also be built by an expression (see 7.40). Lexical structure MIN_VALUE (double, integer)<exp> MIN_VALUE (float, integer)<exp> MIN_VALUE (integer, integer)<exp> MIN_VALUE (unsigned integer, integer)<exp> MIN_VALUE (expression)<exp>
c	n	An increasing integer starting with one, which identifies the MIN_VALUE if more than one is present.
o	SCALING_FACTOR	Purpose The SCALING_FACTOR attribute indicates that the displayed value of the VARIABLE is not the value returned by a device. The displayed value is the value returned by a device multiplied by a factor. Therefore, the EDDL processor shall multiply the value of the VARIABLE returned by a device with its SCALING_FACTOR before it is displayed (or before it is used in any other way). The SCALING_FACTOR shall not equal zero. The SCALING_FACTOR can also be built by an expression (see 7.40). The SCALING_FACTOR is applied for display purposes only; the value of the variable, when assigned to a method local variable, is the unscaled value returned by the device; minimum and maximum values, when used in conjunction with scaling factor, are also unscaled values. Lexical structure SCALING_FACTOR (double)<exp> SCALING_FACTOR (expression)<exp>
o	size	The size of the data type in octets. Size is an integer constant greater than zero and has no upper bound. This value is optional. The default is 1.
o	value	A constant that specifies the value. The type depends on the data type specified with the TYPE attribute which is optional, but if defined a description attribute is required. Equal values are not allowed.

7.33.2.2.2.2 DATE, DATE_AND_TIME, DURATION, TIME, TIME_VALUE

Purpose

VARIABLEs of data type DATE, DATE_AND_TIME, DURATION, TIME, TIME_VALUE are used to specify dates, times, dates and times, and time differences:

- the data type DATE consists of a calendar date;
- the data type DATE_AND_TIME consists of a calendar date and a time;
- the data type DURATION is a time difference which consists of a time in ms and a day count;
- the data type TIME consists of a time and an optional date;
- the data type TIME_VALUE is used to represent time differences or time of day in the required precision for application clock synchronization.

Lexical structure — DATE

DATE specifies that VARIABLE is an integer of three octets. The coding of the octets is specified in D.5.2.

DATE [DEFAULT_VALUE, INITIAL_VALUE]

Lexical structure — DATE_AND_TIME

DATE_AND_TIME specifies that VARIABLE is an integer of seven octets. The coding of the octets is specified in D.5.3.

DATE_AND_TIME [DEFAULT_VALUE, INITIAL_VALUE]

Lexical structure — DURATION

DURATION specifies that VARIABLE is an integer of 6 octets. The first four bit shall have the value zero. Bit 0 to 27 represents the time in ms. The number of days is a 16-bit binary value. The coding of the octets is specified in D.5.4.

DURATION [DEFAULT_VALUE, INITIAL_VALUE]

Lexical structure — TIME

TIME specifies that VARIABLE is an integer of six octets. The time is stated in ms starting at midnight. At midnight, time counting starts with zero.

The date is stated in days relatively to 1984-01-01. On the first day of January 1984, date counting starts with the value zero. The coding of the octets is specified in D.5.5.

TIME [DEFAULT_VALUE, INITIAL_VALUE]

Lexical structure — TIME_VALUE

TIME_VALUE specifies that VARIABLE is an integer of eight octets. It is a fixed point number, which is the time in multiples of 1/32 ms. The coding of the octets is specified in D.5.6.

TIME_VALUE [DEFAULT_VALUE, DISPLAY_FORMAT, EDIT_FORMAT, INITIAL_VALUE, size, TIME_FORMAT, TIME_SCALE]

The attributes of the DATE, DATE_AND_TIME, DURATION, TIME, TIME_VALUE types are specified in Table 166.

Table 166 – DATE, DATE_AND_TIME, DURATION, TIME, TIME_VALUE attributes

Usage	Attribute	Description
o	DEFAULT_VALUE	Purpose The DEFAULT_VALUE attribute specifies the default setting for the VARIABLE. The DEFAULT_VALUE is available for all EDDL types. If no INITIAL_VALUE is defined, the DEFAULT_VALUE is used as an INITIAL_VALUE. The DEFAULT_VALUE can also be built by an expression (see 7.40). Lexical structure DEFAULT_VALUE (integer)<exp> DEFAULT_VALUE (unsigned_integer)<exp> DEFAULT_VALUE (expression)<exp>
o	DISPLAY_FORMAT	See Table 165.
o	EDIT_FORMAT	See Table 165.
o	INITIAL_VALUE	Purpose When a device is instantiated for the first time, the INITIAL_VALUE of a VARIABLE is displayed. Often the DEFAULT_VALUE is conditioned and depends on other parameter settings. In this case the INITIAL_VALUE is used to reconstitute the default settings. The INITIAL_VALUE is available for all types and shall not be conditioned. Lexical structure INITIAL_VALUE (integer)

Usage	Attribute	Description
		INITIAL_VALUE (unsigned_integer)
o	size	The size of the data type in octets. Size shall be either 4 or 8. The default is 4. This attribute is only applicable to TIME_VALUE
o	TIME_FORMAT	Purpose The TIME_FORMAT attribute specifies how the VARIABLE is displayed and edited. The TIME_FORMAT strings may include the %H, %I, %M, %p, %R, %S, and %T format specifiers specified by the ANSI C (see ISO/IEC 9899) function strftime. When TIME_FORMAT is specified, DISPLAY_FORMAT and EDIT_FORMAT shall not be specified. Lexical structure TIME_FORMAT (string)<exp>
o	TIME_SCALE	Purpose The TIME_SCALE attribute specifies how the VARIABLE is scaled, which may be SECONDS, MINUTES, or HOURS. When present, TIME_SCALE infers a scaling factor of 1/32000, 1/1920000, or 1/115200000 and a constant unit of seconds ("s"), minutes ("min"), or hours ("h"), respectively. When TIME_SCALE is specified, DISPLAY_FORMAT and EDIT_FORMAT may also be specified. This attribute is only applicable to TIME_VALUE Lexical structure TIME_SCALE (SECONDS, MINUTES, HOURS)<exp>

7.33.2.2.2.3 BIT_ENUMERATED

Purpose

Variables of data type BIT_ENUMERATED are unsigned integer constants which identifies a bit of the VARIABLE value with its position.

Lexical structure

BIT_ENUMERATED [(description, value, actions, function, help, status-class*)<exp>]+, [DEFAULT_VALUE, INITIAL_VALUE, size]

The attributes of the BIT_ENUMERATED type are specified in Table 167.

Table 167 – BIT_ENUMERATED attributes

Usage	Attribute	Description
m	description	See Table 165.
m	value	An unsigned integer that specifies a bit position. If the value of the VARIABLE supplies at the specified bit positions a logical one, the associated description is displayed. Equal values are not allowed.
o	actions	Actions that will be executed when the bit is set.
o	function	The functional class of the bit. The functional class of a bit is the same as the CLASS of a VARIABLE. If no function is specified, the value of the function class defaults to the class of the variable. Therefore, if all the bits have the same function, then the CLASS of the VARIABLE shall be specified.
o	help	See Table 165.
o	size	See Table 165.
o	status-class	The meaning of the bit. A status bit may belong to more than one status-class (see Table 168). If the VARIABLE is not a status octet, then the bits do not have status classes.
o	DEFAULT_VALUE	Purpose The DEFAULT_VALUE attribute specifies the default setting for the VARIABLE. The DEFAULT_VALUE is available for all EDDL types. If no INITIAL_VALUE is defined, the DEFAULT_VALUE is used as a INITIAL_VALUE. The DEFAULT_VALUE can also be built by an expression (see 7.40). Lexical structure DEFAULT_VALUE (integer)<exp> DEFAULT_VALUE (unsigned_integer)<exp>

Usage	Attribute	Description
		DEFAULT_VALUE (expression)<exp>
o	INITIAL_VALUE	Purpose When a device is instantiated for the first time, the INITIAL_VALUE of a VARIABLE is displayed. Often the DEFAULT_VALUE is conditioned and depends on other parameter settings. In this case the INITIAL_VALUE is used to reconstitute the default settings. The INITIAL_VALUE is available for all types and shall not be conditioned. Lexical structure INITIAL_VALUE (integer) INITIAL_VALUE (unsigned_integer)

Table 168 shows the status-classes for a VARIABLE of type BIT_ENUMERATED.

Table 168 – status-class attributes

Usage	Attribute	Description
m	DATA	An invalid data configuration.
m	HARDWARE	A hardware failure.
m	MISC	A miscellaneous condition.
m	MODE	The device is in a particular mode.
m	PROCESS	A problem with the process connected to the field device.
m	SOFTWARE	A software failure.
m	CORRECTABLE	The bit can be cleared by the EDD application or the connected process.
m	SELF_CORRECTING	The bit will be reset without further intervention.
m	UNCORRECTABLE	The bit is neither self-correcting nor correctable.
m	EVENT	A one-time event.
m	STATE	The device is in a particular state.
m	COMM_ERROR	A communications failure in the other device.
o	IGNORE_IN_HANDHELD	A bit should be ignored in hand-held communicators.
o	IGNORE_IN_TEMPORARY_MASTER	A bit should be ignored in temporary master devices.
m	MORE	There is additional status information available from the device.
o	ALL	Impact all the outputs (see Table 169).
o	AO	Impacts one of the analog outputs (see Table 169).
m	BAD	The output is unreliable and should not be used for control (see Table 169).
o	DV	Impacts one of the dynamic variables (see Table 169).
o	TV	Impacts one of the transmitter variables (see Table 169).
m	DETAIL	The bit is summarized elsewhere in a bit of class summary.
m	SUMMARY	The bit is a logical combination of other bits of class detail.
o	INFO	The bit provides information about the status of the process or device.
o	WARNING	The bit provides a warning about the status of the process or device.
o	IGNORE_IN_HOST	The bit shall be ignored in fixed, permanently attached host.

The status-class ALL, AO, DV and TV provides more status information about the output of a device.

Lexical structure — ALL

ALL [AUTO_BAD, AUTO_GOOD, MANUAL_BAD, MANUAL_GOOD]

Lexical structure — AO

AO [AUTO_BAD, AUTO_GOOD, integer, MANUAL_BAD, MANUAL_GOOD]

Lexical structure — DV

DV [AUTO_BAD, AUTO_GOOD, integer, MANUAL_BAD, MANUAL_GOOD]

Lexical structure — TV

TV [AUTO_BAD, AUTO_GOOD, integer, MANUAL_BAD, MANUAL_GOOD]

The attributes are specified in Table 169.

Table 169 – ALL, AO, DV, TV attributes

Usage	Attribute	Description
s	AUTO_BAD	The value is coming from an application process and is invalid.
s	AUTO_GOOD	The value is coming from an application process and is invalid.
m	integer	The VARIABLE value coming from the device.
s	MANUAL_BAD	The value is not coming from an application process and is invalid.
s	MANUAL_GOOD	The value is not coming from an application process and is invalid.

7.33.2.2.2.4 ENUMERATED

Purpose

A VARIABLE of type ENUMERATED is an unsigned integer. An associated list of triplets (integer, description, help) specifies the possible values of the VARIABLE plus a description and an optional help.

Lexical structure

ENUMERATED [(description, value, help)<exp>]+, [DEFAULT_VALUE, INITIAL_VALUE, size]

The attributes of the ENUMERATED type are specified in Table 170.

Table 170 – Enumerated types attributes

Usage	Attribute	Description
m	description	See Table 165.
m	value	A constant that specifies the VARIABLE value. The type is an unsigned integer. The enumeration list is optional but if defined a value and a description is required. Equal values are not allowed.
o	DEFAULT_VALUE	Purpose The DEFAULT_VALUE specifies the default setting for the VARIABLE. The DEFAULT_VALUE is available for all EDDL types. If no INITIAL_VALUE is defined, the DEFAULT_VALUE is used as a INITIAL_VALUE. The DEFAULT_VALUE can also be built by an expression, see 7.40). Lexical structure DEFAULT_VALUE (unsigned_integer)<exp> DEFAULT_VALUE (expression)<exp>
o	help	See Table 165.
o	INITIAL_VALUE	Purpose When a device is instantiated for the first time, the INITIAL_VALUE of a VARIABLE is displayed. Often the DEFAULT_VALUE is conditioned and depends on other parameter settings. In this case the INITIAL_VALUE is used to reconstitute the default settings. The INITIAL_VALUE is available for all types and shall not be conditioned. Lexical structure INITIAL_VALUE (unsigned_integer)
o	size	See Table 165.

7.33.2.2.2.5 INDEX

Purpose

A VARIABLE of TYPE INDEX is an unsigned integer which is used to reference the elements of a REFERENCE_ARRAY (see 7.27). The integer attribute of ELEMENTS (see 7.27.2) specifies the allowable values of the VARIABLE.

NOTE When an INDEX VARIABLE is presented to the user, the text associated with the indices of the array shall be displayed, not the numeric value of the VARIABLE.

Lexical structure

INDEX (reference, size)<exp>

The attributes of the index type are specified in Table 171.

Table 171 – Index type attributes

Usage	Attribute	Description
o	size	The size of the VARIABLE in octets. This value is an integer constant greater than zero and has no upper bound. The default is 1. The REFERENCE_ARRAY shall contain only elements, which does not exceeds the index size of the variable.
m	reference	A reference to a REFERENCE_ARRAY instance.

7.33.2.2.2.6 Object reference types

Purpose

VARIABLEs of the type OBJECT_REFERENCE allow the access to values of other devices.

Lexical structure

OBJECT_REFERENCE DEFAULT_REFERENCE

The attributes of the OBJECT_REFERENCE types are specified in Table 172.

Table 172 – Object reference type attribute

Usage	Attribute	Description
o	DEFAULT_REFERENCE	The VARIABLE is used to access the values of other devices. If no DEFAULT_REFERENCE is defined the VARIABLE is initialized with SELF (see Table 173).

The attributes of the DEFAULT-REFERENCE are specified in Table 173.

Table 173 – DEFAULT_REFERENCE attributes

Usage	Attribute	Description
s	CHILD	Navigates one hierarchy downward.
s	FIRST	Navigates to the first object in the hierarchy.
s	LAST	Navigates to the last object in the hierarchy.
s	NEXT	Navigates to the next object of the same hierarchy.
s	NULL	An empty object reference.
s	PARENT	Navigates one hierarchy bottom-up.
s	PREV	Navigates to the previous object of the same hierarchy.
s	reference	A reference to a device projected in the network.
s	SELF	Navigates back to themselves in their own hierarchy.

7.33.2.2.2.7 ASCII, BIT_STRING, EUC, PACKED_ASCII, OCTET, PASSWORD, VISIBLE

Purpose

String variable types include the following.

- ASCII – is for specifying a sequence of characters from the ISO Latin-1 character set.
- BITSTRING – Variables of data type BIT_ENUMERATED are string constants which identify the position of a character of the VARIABLE value with its position.
- EUC (Extended Unit code) – is the internal code for specific characters (for example, China: EUC-CN, Taiwan EUC-TW). EUC is used to handle East Asian languages (ISO/IEC 10646-1).
- The EDDL processor shall support an encoding scheme where characters are encoded using a variable number of octets. Typically certain octets signal the beginning of a character and how many additional octets are used to encode the character. Character sets with a large number of elements are often stored using a packing scheme.
- PACKED_ASCII – is a subset of ASCII produced by removing the two most significant bits of each ASCII character (see D.5.7).
- PASSWORD – is a string type and is intended for specifying password strings. Except for how the variable is presented to the user, password and ASCII string types are identical.
- VISIBLE – This string is an ordered sequence of characters from ISO/IEC 10646-1 and ISO/IEC 2375 character set.
- OCTET – is for specifying a sequence of unformatted binary data whose definition is determined by the implementation of the device.

Lexical structure — EUC

EUC [(string, description, help)<exp>]+, [size, DEFAULT_VALUE, INITIAL_VALUE]

Lexical structure — ASCII

ASCII [(string, description, help)<exp>]+, [size, DEFAULT_VALUE, INITIAL_VALUE]

Lexical structure — PACKED_ASCII

PACKED_ASCII [(string, description, help)<exp>]+, [size, DEFAULT_VALUE, INITIAL_VALUE]

Lexical structure — PASSWORD

PASSWORD [(string, description, help)<exp>]+, [size, DEFAULT_VALUE, INITIAL_VALUE]

Lexical structure — BITSTRING

BITSTRING [(string, description, help)<exp>]+, [size, DEFAULT_VALUE, INITIAL_VALUE]

Lexical structure — VISIBLE

VISIBLE [(string, description, help)<exp>]+, [size, DEFAULT_VALUE, INITIAL_VALUE]

Lexical structure — OCTET

OCTET [(string, description, help)<exp>]+, [size, DEFAULT_VALUE, INITIAL_VALUE]

The attributes of the string types are specified in Table 174.

Table 174 – String types attributes

Usage	Attribute	Description
o	size	The size of the data type in octets. Size is an integer constant greater than zero and has no upper bound. This value is not optional; a size must be specified.
o	DEFAULT_VALUE	Purpose The DEFAULT_VALUE specifies the default setting for the VARIABLE. The DEFAULT_VALUE is available for all EDDL types. If no INITIAL_VALUE is defined, the DEFAULT_VALUE is used as a INITIAL_VALUE. Lexical structure DEFAULT_VALUE (string)<exp>
o	INITIAL_VALUE	Purpose When a device is instantiated for the first time, the INITIAL_VALUE of a VARIABLE is displayed. Often the DEFAULT_VALUE is conditioned and depends on other parameter settings. In this case the INITIAL_VALUE is used to reconstitute the default settings. The INITIAL_VALUE is available for all types and shall not be conditioned. Lexical structure INITIAL_VALUE (string)
o	string	A string constant. The enumeration list is optional but if defined a string and a description is required. Equal strings are not allowed.
o	description	See Table 165.
o	help	See Table 165.

7.33.2.3 CONSTANT_UNIT**Purpose**

The CONSTANT_UNIT attribute is used, if a VARIABLE has a units code which never changes. The CONSTANT_UNIT is specified as a string that will be displayed along with the value. A VARIABLE without a constant unit either has no units associated with it or the units are not constant.

Lexical structure

CONSTANT_UNIT (string)<exp>

The attribute of CONSTANT_UNIT is specified in Table 175.

Table 175 – CONSTANT_UNIT attribute

Usage	Attribute	Description
m	string	The units code string to be displayed.

7.33.2.4 DEFAULT_VALUE**Purpose**

The DEFAULT_VALUE attribute specifies the default setting for the VARIABLE. If no INITIAL_VALUE is defined, the DEFAULT_VALUE is used as the INITIAL_VALUE.

Lexical structure

DEFAULT_VALUE (expression)<exp>

NOTE 1 The type of the expression must be consistent with the type of the VARIABLE.

NOTE 2 The DEFAULT_VALUE may also be specified via the TYPE attribute of the VARIABLE.

The attribute of DEFAULT_VALUE is specified in Table 176.

Table 176 – DEFAULT_VALUE attribute

Usage	Attribute	Description
m	expression	The default value of the variable.

7.33.2.5 INITIAL_VALUE**Purpose**

The INITIAL_VALUE attribute specifies the value of a VARIABLE that is displayed when a device is instantiated for the first time. Often the DEFAULT_VALUE is conditioned and depends on other parameter settings. In this case, the INITIAL_VALUE is used to reconstitute the default settings. The INITIAL_VALUE is available for all types and shall not be conditioned.

Lexical structure

INITIAL_VALUE expression

NOTE 1 The type of the expression must be consistent with the type of the VARIABLE.

NOTE 2 The INITIAL_VALUE may also be specified via the TYPE attribute of the VARIABLE.

The attribute of INITIAL_VALUE is specified in Table 177.

Table 177 – INITIAL_VALUE attribute

Usage	Attribute	Description
m	expression	The initial value of the variable.

7.33.2.6 POST_EDIT_ACTIONS**Purpose**

The POST_EDIT_ACTIONS attribute specifies METHODS that shall be executed after the user has finished editing the VARIABLE. The specified METHODS shall be executed in the order they appear. If a METHOD exits for an unplanned reason, the following METHODS are not executed.

The POST_EDIT_ACTIONS attribute shall not be expressed using a conditional (IF, IF-ELSE, SELECT). If conditional behavior is required, the methods invoked by the action shall accommodate it.

Lexical structure

POST_EDIT_ACTIONS [reference]+, DEFINITION

The attributes of POST_EDIT_ACTIONS are specified in Table 178.

**Table 178 – POST_EDIT_ACTIONS, PRE_EDIT_ACTIONS, POST_READ_ACTIONS,
PRE_READ_ACTIONS, POST_WRITE_ACTIONS, PRE_WRITE_ACTIONS,
REFRESH_ACTIONS attributes**

Usage	Attribute	Description
o	reference	A reference to a METHOD instance or a METHOD instance with arguments.
o	DEFINITION	Lexical element (see 7.36.1).

7.33.2.7 POST_READ_ACTIONS**Purpose**

The POST_READ_ACTIONS attribute specifies METHODS that shall be executed after the VARIABLE was read from the device. The specified METHODS shall be executed in the order they appear. If a METHOD exits for an unplanned reason, the following METHODS are not executed.

The POST_READ_ACTIONS attribute shall not be expressed using a conditional (IF, IF-ELSE, SELECT). If conditional behavior is required, the methods invoked by the action shall accommodate it.

Lexical structure

POST_READ_ACTIONS [reference]+, DEFINITION

The attributes of POST_READ_ACTIONS are specified in Table 178.

7.33.2.8 POST_WRITE_ACTIONS

Purpose

The POST_WRITE_ACTIONS attribute specifies METHODS that shall be executed after the VARIABLE has been written to the device. The specified METHODS shall be executed in the order they appear. If a METHOD exits for an unplanned reason, the following METHODS are not executed.

The POST_WRITE_ACTIONS attribute shall not be expressed using a conditional (IF, IF-ELSE, SELECT). If conditional behavior is required, the methods invoked by the action shall accommodate it.

Lexical structure

POST_WRITE_ACTIONS [reference]+, DEFINITION

The attributes of POST_WRITE_ACTIONS are specified in Table 178.

7.33.2.9 PRE_EDIT_ACTIONS

Purpose

The PRE_EDIT_ACTIONS attribute specifies METHODS that shall be executed immediately when the VARIABLE is going to be edited. The specified METHODS shall be executed in the order they appear. If a METHOD exits for an unplanned reason, the following METHODS are not executed.

The PRE_EDIT_ACTIONS attribute shall not be expressed using a conditional (IF, IF-ELSE, SELECT). If conditional behavior is required, the methods invoked by the action shall accommodate it.

Lexical structure

PRE_EDIT_ACTIONS [reference]+, DEFINITION

The attributes of PRE_EDIT_ACTIONS are specified in Table 178.

7.33.2.10 PRE_READ_ACTIONS

Purpose

The PRE_READ_ACTIONS attribute specifies METHODS that shall be executed before the VARIABLE is read. The specified METHODS shall be executed in the order they appear. If a METHOD exits for an unplanned reason, the following METHODS are not executed.

The PRE_READ_ACTIONS attribute shall not be expressed using a conditional (IF, IF-ELSE, SELECT). If conditional behavior is required, the methods invoked by the action shall accommodate it.

Lexical structure

PRE_READ_ACTIONS [reference]+, DEFINITION

The attributes of PRE_READ_ACTIONS are specified in Table 178.

7.33.2.11 PRE_WRITE_ACTIONS

Purpose

The PRE_WRITE_ACTIONS attribute specifies METHODS that shall be executed before the VARIABLE is written to the device. The specified METHODS shall be executed in the order

they appear. If a METHOD exits for an unplanned reason, the following METHODS are not executed.

The PRE_WRITE_ACTIONS attribute shall not be expressed using a conditional (IF, IF-ELSE, SELECT). If conditional behavior is required, the methods invoked by the action shall accommodate it.

Lexical structure

PRE_WRITE_ACTIONS [reference]+, DEFINITION

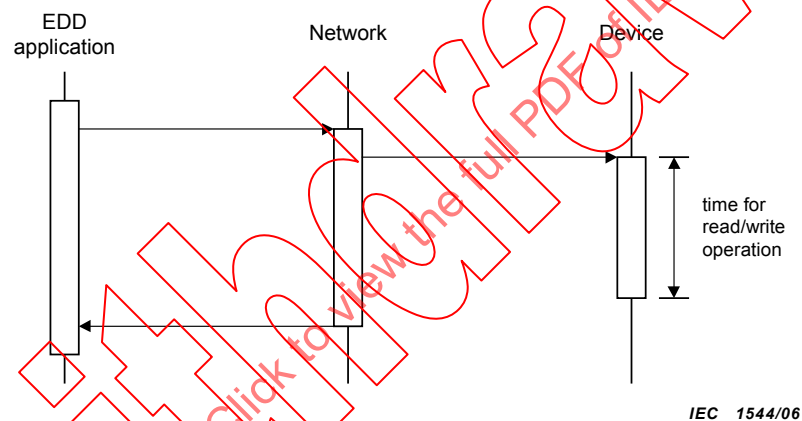
The attributes of PRE_WRITE_ACTIONS are specified in Table 178.

7.33.2.12 READ_TIMEOUT, WRITE_TIMEOUT

Purpose

The READ_TIMEOUT attribute specifies the length of time, in ms, the EDD application shall wait for the returned VARIABLE. Similarly, the WRITE_TIMEOUT attribute specifies the length of time, in ms, an EDD application shall wait for confirmation that the VARIABLE is successfully written to the device.

For example, a WRITE_TIMEOUT should indicate the length of time it takes if a device stores the value of a variable.



IEC 1544/06

Figure 29 – Time for read-and-write operation

Lexical structure

READ_TIMEOUT (integer) <exp>

WRITE_TIMEOUT (integer) <exp>

The attribute of READ_TIMEOUT and WRITE_TIMEOUT is specified in Table 179.

Table 179 – READ/WRITE_TIMEOUT attributes

Usage	Attribute	Description
m	integer	The time period in ms that the EDD application waits for a device to complete its requested update.

7.33.2.13 REFRESH_ACTIONS

Purpose

The REFRESH_ACTIONS attribute specifies METHODS that shall be executed whenever the VARIABLE is displayed or refreshed. The specified METHODS shall be executed in the order they appear. If a METHOD exits for an unplanned reason, the following METHODS are not executed.

The REFRESH_ACTIONS attribute shall not be expressed using a conditional (IF, IF-ELSE, SELECT). If conditional behavior is required, the methods invoked by the action shall accommodate it.

Lexical structure

```
REFRESH_ACTIONS [reference]+, DEFINITION
```

The attributes of REFRESH_ACTIONS are specified in Table 178.

7.33.2.14 STYLE**Purpose**

The STYLE attribute specifies the way a VARIABLE is displayed. It is an additional information which is individually interpreted by the EDDL application.

Lexical structure

```
STYLE (string)<exp>
```

The attribute of STYLE is specified in Table 180.

Table 180 – STYLE attribute

Usage	Attribute	Description
m	string	A string which is used for display purposes.

7.34 VARIABLE_LIST**Purpose**

A VARIABLE_LIST is a group of EDD communication objects (VARIABLE, VALUE_ARRAY or RECORDS). VARIABLE_LISTS are used to group objects for application convenience.

NOTE A VARIABLE_LIST may be transferred as one communication object.

Lexical structure

```
VARIABLE_LIST identifier, [MEMBERS, HELP, LABEL, RESPONSE_CODES]
```

The attributes are specified in Table 181.

Table 181 – VARIABLE_LIST attributes

Usage	Attribute	Description
m	MEMBERS	Lexical element (see 7.36.12).
o	HELP	Lexical element (see 7.36.8).
o	LABEL	Lexical element (see 7.36.9).
o	RESPONSE_CODES	Lexical element (see 7.36.14).

7.35 WAVEFORM**7.35.1 General structure****Purpose**

A WAVEFORM describes a data set that may be displayed by a GRAPH.

Lexical structure

```
WAVEFORM identifier [TYPE, EMPHASIS, EXIT_ACTIONS, HANDLING, HELP, INIT_ACTIONS, KEY_POINTS, LABEL, LINE_COLOR, LINE_TYPE, REFRESH_ACTIONS, VALIDITY, Y_AXIS]
```

The attributes of WAVEFORM are specified in Table 182.

Table 182 – WAVEFORM attributes

Usage	Attribute	Description
m	TYPE	Lexical element (see 7.35.2.1).
o	EMPHASIS	Lexical element (see 7.36.5).

Usage	Attribute	Description
o	EXIT_ACTIONS	Lexical element (see 7.35.2.2).
o	HANDLING	Lexical element (see 7.36.6).
o	HELP	Lexical element (see 7.36.8).
o	INIT_ACTIONS	Lexical element (see 7.35.2.3).
o	KEY_POINTS	Lexical element (see 7.35.2.4).
o	LABEL	Lexical element (see 7.36.9).
o	LINE_COLOR	Lexical element (see 7.36.10).
o	LINE_TYPE	Lexical element (see 7.36.11).
o	REFRESH_ACTIONS	Lexical element (see 7.35.2.5).
o	VALIDITY	Lexical element (see 7.36.16).
o	Y_AXIS	Lexical element (see 7.35.2.6).

7.35.2 Specific attributes

7.35.2.1 TYPE

7.35.2.1.1 General structure

Purpose

The TYPE attribute specifies the type of data contained in the WAVEFORM.

Lexical structure

TYPE [XY, YT, HORIZONTAL, VERTICAL]

The attributes of TYPE are specified in Table 183.

Table 183 – TYPE attribute

Usage	Attribute	Description
s	XY	Lexical element (see 7.35.2.1.2.1).
s	YT	Lexical element (see 7.35.2.1.2.2).
s	HORIZONTAL	Lexical element (see 7.35.2.1.2.3).
s	VERTICAL	Lexical element (see 7.35.2.1.2.4).

7.35.2.1.2 Specific attributes

7.35.2.1.2.1 XY

Purpose

The XY attribute specifies a WAVEFORM that contains a list of (x,y) points.

Lexical structure

XY [X_VALUES, Y_VALUES, NUMBER_OF_POINTS]

The attributes of XY are specified in Table 184.

Table 184 – XY attribute

Usage	Attribute	Description
m	X_VALUES	Purpose The X_VALUES attribute specifies the X coordinate of each point in the WAVEFORM. For each X coordinate specified in X_VALUES there shall be a corresponding Y coordinate specified in Y_VALUES Lexical Structure X_VALUES [reference, double, float, integer, unsigned_integer]+ The attributes of X_VALUES are specified in Table 190.

Usage	Attribute	Description
m	Y_VALUES	Purpose The Y_VALUES attribute specifies the Y coordinate of each point in the WAVEFORM. For each Y coordinate specified in Y_VALUES there shall be a corresponding X coordinate specified in X_VALUES Lexical Structure Y_VALUES [(reference, double, float, integer, unsigned_integer)+ The attributes of Y_VALUES are specified in Table 190.
o	NUMBER_OF_POINTS	Purpose The NUMBER_OF_POINTS attribute specifies the number of valid data points in X_VALUES and Y_VALUES. By default, the number of points in the WAVEFORM without a NUMBER_OF_POINTS attribute equals the size of X_VALUES and Y_VALUES Lexical Structure NUMBER_OF_POINTS (integer)<exp>

7.35.2.1.2.2 YT

Purpose

The YT attribute specifies a WAVEFORM that contains a list of points that are defined via an initial X coordinate, a X increment between successive points and a list of Y values.

Lexical structure

YT [X_INITIAL, X_INCREMENT, Y_VALUES, NUMBER_OF_POINTS]

The attributes of YT are specified in Table 185.

Table 185 – YT attribute

Usage	Attribute	Description
m	X_INITIAL	Purpose The X_INITIAL attribute specifies the X coordinate of the first point in the WAVEFORM. Lexical Structure X_INITIAL (reference)<exp> X_INITIAL (double)<exp> X_INITIAL (float)<exp> X_INITIAL (integer)<exp> X_INITIAL (unsigned_integer)<exp> The reference is a reference to either a VARIABLE instance, a member of a RECORD instance, an element of a VALUE_ARRAY instance or an element of a REFERENCE_ARRAY instance.
m	X_INCREMENT	Purpose The X_INCREMENT attribute specifies difference between the X coordinates of adjacent points in the WAVEFORM. Lexical Structure X_INCREMENT (reference)<exp> X_INCREMENT (double)<exp> X_INCREMENT (float)<exp> X_INCREMENT (integer)<exp> X_INCREMENT (unsigned_integer)<exp> The reference is a reference to either a VARIABLE instance, a member of a RECORD instance, an element of a VALUE_ARRAY instance or an element of a REFERENCE_ARRAY instance.
m	Y_VALUES	Purpose The Y_VALUES attribute specifies the Y coordinate of each point in the WAVEFORM. Lexical Structure Y_VALUES [(reference, double, float, integer, unsigned_integer)<exp>]+ The attributes of Y_VALUES are specified in Table 190.
o	NUMBER_OF_POINTS	Purpose The NUMBER_OF_POINTS attribute specifies the number of valid data points in X_VALUES and Y_VALUES. By default, the number of points in the WAVEFORM without a NUMBER_OF_POINTS attribute equals the size of X_VALUES and Y_VALUES.

Usage	Attribute	Description
		Lexical Structure NUMBER_OF_POINTS (integer)<exp>

7.35.2.1.2.3 HORIZONTAL

Purpose

The HORIZONTAL attribute specifies a WAVEFORM that contains horizontal lines.

Lexical structure

HORIZONTAL Y_VALUES

The attributes of HORIZONTAL are specified in Table 186.

Table 186 – HORIZONTAL attribute

Usage	Attribute	Description
m	Y_VALUES	Purpose The Y_VALUES attribute specifies the Y coordinate of each horizontal line in the WAVEFORM. Lexical Structure Y_VALUES [reference, double, float, integer, unsigned_integer]+ The attributes of Y_VALUES are specified in Table 190.

7.35.2.1.2.4 VERTICAL

Purpose

The VERTICAL attribute specifies a WAVEFORM that contains vertical lines.

Lexical structure

VERTICAL X_VALUES

The attributes of VERTICAL are specified in Table 187.

Table 187 – VERTICAL attribute

Usage	Attribute	Description
m	X_VALUES	Purpose The X_VALUES attribute specifies the X coordinate of each vertical line in the WAVEFORM. Lexical Structure X_VALUES [reference, double, float, integer, unsigned_integer]+ The attributes of X_VALUES are specified in Table 190.

7.35.2.2 EXIT_ACTIONS

Purpose

The EXIT_ACTIONS attribute specifies actions that are executed when the GRAPH containing the WAVEFORM is closed.

The specified METHODS shall be executed in the order they appear. If a METHOD exits for an unplanned reason, the remaining METHODS shall not be executed.

The EXIT_ACTIONS attribute shall not be expressed using a conditional (IF, IF-ELSE, SELECT). If conditional behavior is required, the methods invoked by the action shall accommodate it.

Lexical structure

EXIT_ACTIONS [reference]+, DEFINITION

The attributes of EXIT_ACTIONS are specified in Table 188.

7.35.2.3 INIT_ACTIONS

Purpose

The INIT_ACTIONS attribute specifies actions that are executed before the WAVEFORM is initially displayed by a GRAPH. These methods are generally used to load the WAVEFORM with data from a device or a FILE.

The specified METHODS shall be executed in the order they appear. If a METHOD exits for an unplanned reason, the remaining METHODS shall not be executed. Furthermore, the EDD application shall not draw the WAVEFORM if a METHOD exits for an unplanned reason.

The INIT_ACTIONS attribute shall not be expressed using a conditional (IF, IF-ELSE, SELECT). If conditional behavior is required, the methods invoked by the action shall accommodate it.

Lexical structure

INIT_ACTIONS [reference]+, DEFINITION

The attributes of INIT_ACTIONS are specified in Table 188.

Table 188 – EXIT_ACTIONS, INIT_ACTIONS, REFRESH_ACTIONS attribute

Usage	Attribute	Description
o	reference	A reference to a METHOD instance or a METHOD instance with arguments.
o	DEFINITION	Lexical element (see 7.36.4).

7.35.2.4 KEY_POINTS

7.35.2.4.1 General structure

Purpose

The KEY_POINTS attribute specifies key points in the WAVEFORM that should be highlighted by the EDD application. The key points need not directly correspond to the data points specified via the TYPE attribute. The way in which these points are highlighted is defined by the EDD specification. By default, a WAVEFORM without a KEY_POINTS attribute should not have any of its points highlighted.

Lexical structure

KEY_POINTS [X_VALUES, Y_VALUES]

The attributes of KEY_POINTS are specified in Table 189.

Table 189 – KEY_POINTS attribute

Usage	Attribute	Description
m	X_VALUES	The X coordinate of each KEY_POINT.
m	Y_VALUES	The Y coordinate of each KEY_POINT.

7.35.2.4.2 Specific attributes

7.35.2.4.2.1 X_VALUES

Purpose

The X_VALUES attribute specifies the X coordinate of each point that is to be highlighted. For each X coordinate specified in X_VALUES there shall be a corresponding Y coordinate specified in Y_VALUES.

Lexical structure

X_VALUES [reference, double, float, integer, unsigned_integer]+

The attributes of X_VALUES are specified in Table 190.

Table 190 – X_VALUES, Y_VALUES attribute

Usage	Attribute	Description
o	reference	A reference to either a VARIABLE instance, a member of a RECORD instance, an element of a VALUE_ARRAY instance, an element of a REFERENCE_ARRAY instance, a VALUE_ARRAY instance or a REFERENCE_ARRAY instance.
o	double	A double-precision floating-point constant.
o	float	A single-precision floating-point constant.
o	integer	An integer constant.
o	unsigned_integer	An unsigned integer constant.

7.35.2.4.2.2 Y_VALUES**Purpose**

The Y_VALUES attribute specifies the Y coordinate of each point that is to be highlighted. For each Y coordinate specified in Y_VALUES, there shall be a corresponding X coordinate specified in X_VALUES.

Lexical structure

Y_VALUES [reference, double, float, integer, unsigned_integer]+

The attributes of Y_VALUES are specified in Table 190.

7.35.2.5 REFRESH_ACTIONS**Purpose**

The REFRESH_ACTIONS attribute specifies actions that are executed whenever changes are made to the X_AXIS or Y_AXIS of the GRAPH in which the WAVEFORM is being displayed.

The specified METHODS shall be executed in the order they appear. If a METHOD exits for an unplanned reason, the remaining METHODS shall not be executed. Furthermore, the X_VALUES and Y_VALUES of the WAVEFORM shall be restored to their original values by the EDD application, and the WAVEFORM on the GRAPH shall not be redrawn.

The REFRESH_ACTIONS attribute shall not be expressed using a conditional (IF, IF-ELSE, SELECT). If conditional behavior is required, the methods invoked by the action shall accommodate it.

Lexical structure

REFRESH_ACTIONS [reference]+, DEFINITION

The attributes of REFRESH_ACTIONS are specified in Table 188.

7.35.2.6 Y_AXIS**Purpose**

The Y_AXIS attribute specifies the way the Y axis of the WAVEFORM is displayed. Each WAVEFORM on a GRAPH share a common X_AXIS but each WAVEFORM on a GRAPH has its own Y_AXIS. By default, the display of the Y axis of a GRAPH without a Y_AXIS attribute is inferred from the data in the WAVEFORM.

Lexical structure

Y_AXIS (reference)<exp>

The attribute of Y_AXIS is specified in Table 191.

Table 191 – Y_AXIS attribute

Usage	Attribute	Description
m	reference	A reference to an AXIS instance.

7.36 Common attributes

7.36.1 CLASSIFICATION

Purpose

The CLASSIFICATION attribute is a single enumerated value that indicates the (possibly multiple) levels classification for this device.

Lexical structure

CLASSIFICATION [see attributes in Table 192]

The attributes of CLASSIFICATION are specified in Table 192.

Table 192 – CLASSIFICATION attribute

Usage	Attribute	Description
s	ACTUATOR	The component represents this.
s	ACTUATOR_ELECTRO_PNEUMATIC	The component represents this.
s	ACTUATOR_ELECTRIC	The component represents this.
s	ACTUATOR_HYDRAULIC	The component represents this.
s	CONVERTER	The component represents this.
s	CONTROLLER	The component represents this.
s	DISCRETE_IN	The component represents this.
s	DISCRETE_OUT	The component represents this.
s	ELECTRICAL_DISTRIBUTION	The component represents this.
s	ELECTRICAL_DISTRIBUTION_POWER_MONITORING	The component represents this.
s	FREQUENCY_CONVERTER	The component represents this.
s	INDICATOR	The component represents this.
s	NETWORK_COMPONENT	The component represents this.
s	NETWORK_COMPONENT_WIRELESSADAPTER	The component represents this.
s	REMOTEIO	The component represents this.
s	RECORDER	The component represents this.
s	SENSOR	The component represents this.
s	SENSOR_ACOUSTIC	The component represents this.
s	SENSOR_ANALYTIC	The component represents this.
s	SENSOR_ANALYTIC_GAS	The component represents this.
s	SENSOR_ANALYTIC_LIQUID	The component represents this.
s	SENSOR_CONCENTRATION	The component represents this.
s	SENSOR_DENSITY	The component represents this.
s	SENSOR_DENSITY_RADIOMETRIC	The component represents this.
s	SENSOR_FLOW	The component represents this.
s	SENSOR_FLOW_CORIOLIS	The component represents this.
s	SENSOR_FLOW_ELECTRO_MAGNETIC	The component represents this.
s	SENSOR_FLOW_MECHANICAL	The component represents this.
s	SENSOR_FLOW_RADIOMETRIC	The component represents this.
s	SENSOR_FLOW_THERMAL	The component represents this.
s	SENSOR_FLOW_ULTRASONIC	The component represents this.
s	SENSOR_FLOW_VORTEX_COUNTER	The component represents this.
s	SENSOR_LEVEL	The component represents this.

Usage	Attribute	Description
s	SENSOR_LEVEL_BUOYANCY	The component represents this.
s	SENSOR_LEVEL_CAPACITIVE	The component represents this.
s	SENSOR_LEVEL_ECHO	The component represents this.
s	SENSOR_LEVEL_HYDROSTATIC	The component represents this.
s	SENSOR_LEVEL_RADIOMETRIC	The component represents this.
s	SENSOR_MULTIVARIABLE	The component represents this.
s	SENSOR_POSITION	The component represents this.
s	SENSOR_PRESSURE	The component represents this.
s	SENSOR_TEMPERATURE	The component represents this.
s	SEMICONDUCTOR	The component represents this.
s	SWITCHGEAR	The component represents this.
s	UNIVERSAL	The component represents this.
s	string	The component represents this.

7.36.2 COMPONENT_PARENT

Purpose

The COMPONENT_PARENT attribute is a reference to a COMPONENT_FOLDER. COMPONENT_PARENT can be used instead of PROTOCOL and CLASSIFICATION to simplify (shorten) the COMPONENT_PATH declaration. If the PROTOCOL and/or CLASSIFICATION is defined in the COMPONENT_PARENT (or its parent) the PROTOCOL and/or CLASSIFICATION shall not be redefined. If the COMPONENT_PARENT is specified, the COMPONENT_PATH shall be defined.

NOTE The COMPONENT_PARENT is predefined by the HCF and FF EDD library structure.

Lexical structure

COMPONENT_PARENT string

The attributes of COMPONENT_PARENT are specified in Table 193.

Table 193 – COMPONENT_PARENT attribute

Usage	Attribute	Description
m	string	The parent component in the catalog.

7.36.3 COMPONENT_PATH

Purpose

The COMPONENT_PATH attribute allows the EDD manufacturer to designate the location in the catalog of an EDD Application. The path only specifies the manufacturer specific part of the location without the PROTOCOL, CLASSIFICATION and MANUFACTURER. If a COMPONENT_PARENT is used the PROTOCOL, CLASSIFICATION and MANUFACTURER may not be defined and the COMPONENT_PATH defines only the sub hierarchy.

NOTE The COMPONENT_PATH is predefined by the HCF and FF EDD library structure.

Lexical structure

COMPONENT_PATH string

The attributes of COMPONENT_PATH are specified in Table 194.

Table 194 – COMPONENT_PATH attribute

Usage	Attribute	Description
m	string	The manufacturer specific path of the component in the catalog.

7.36.4 DEFINITION

Purpose

The DEFINITION attribute specifies actions to be performed and is based on a procedural programming language (for example, ANSI C). In the DEFINITIONS, METHODS and Builtins (see Annex B) may be called. Within a DEFINITION EDD basic elements can be used.

The following items shall be supported.

- Basic data types (see D.5.1)
- Arrays
- Arithmetic operators
- Statements
- Invocation of METHODS and Builtins
- METHODS with arguments (call by value, call by reference)

Call by value is used to pass a copy of the data to the METHOD. Changes to the copy within the METHOD do not change the original data.

Call by reference is used to access data directly. Changes effect the original data.

Lexical structure

DEFINITION compound-statement

The attribute of DEFINITION is specified in Table 195.

Table 195 – DEFINITION attributes

Usage	Attribute	Description
o	compound-statement	A block of instructions for a supported procedural programming language.

7.36.5 EMPHASIS

Purpose

The EMPHASIS attribute specifies whether or not a user interface element should be emphasized when it is displayed. The way in which it is emphasized is defined by the EDD application. By default, a user interface element without an EMPHASIS attribute should not be emphasized.

Lexical structure

EMPHASIS (FALSE, TRUE) <exp>

The attributes of EMPHASIS are specified in Table 196.

Table 196 – EMPHASIS attribute

Usage	Attribute	Description
s	FALSE	The user interface element should not be emphasized.
s	TRUE	The user interface element should be emphasized.

7.36.6 HANDLING

Purpose

The HANDLING attribute specifies the operations that can be performed on an element. By default, an element without a HANDLING attribute can be read and written.

Lexical structure

HANDLING (READ, READ_WRITE, WRITE)<exp>

The attributes of HANDLING are specified in Table 197.

Table 197 – HANDLING attribute

Usage	Attribute	Description
s	READ	The element may only be read.
s	READ_WRITE	The element may be read and written.
s	WRITE	The element may be written.

7.36.7 HEIGHT

Purpose

The HEIGHT attribute specifies the vertical size of a user interface element when it is displayed. By default, the height of a user interface element without a HEIGHT attribute is MEDIUM.

Lexical structure

HEIGHT (XX_SMALL, X_SMALL, SMALL, MEDIUM, LARGE, X_LARGE, XX_LARGE)

The attributes of HEIGHT are specified in Table 198.

Table 198 – HEIGHT/WIDTH attribute

Usage	Attribute	Description
s	XX_SMALL	The user interface element should occupy an extra, extra small portion of the window in which it is being displayed.
s	X_SMALL	The user interface element should occupy an extra small portion of the window in which it is being displayed.
s	SMALL	The user interface element should occupy a small portion of the window in which it is being displayed.
s	MEDIUM	The user interface element should occupy a medium portion of the window in which it is being displayed.
s	LARGE	The user interface element should occupy a large portion of the window in which it is being displayed.
s	X_LARGE	The user interface element should occupy an extra large portion of the window in which it is being displayed.
s	XX_LARGE	The user interface element should occupy an extra, extra large portion of the window in which it is being displayed.

7.36.8 HELP

Purpose

The HELP attribute specifies text, which provides a description of the construct.

NOTE This text is intended to be used by EDD applications for user information.

Lexical structure

HELP (string)<exp>

The attribute of HELP is specified in Table 199.

Table 199 – HELP attribute

Usage	Attribute	Description
m	string	User readable text.

7.36.9 LABEL

Purpose

The LABEL attribute specifies the displayed designation of an element.

Lexical structure

LABEL (string)<exp>

The attribute of LABEL is specified in Table 200.

Table 200 – LABEL attribute

Usage	Attribute	Description
m	string	User readable text.

7.36.10 LINE_COLOR

Purpose

The LINE_COLOR attribute specifies the color that should be used to display a user interface element. By default, a user interface element without a LINE_COLOR attribute will be displayed using a default color defined by the EDD application.

Lexical structure

LINE_COLOR (integer, reference)<exp>

Colors are specified via a RGB value, as defined in Cascading Style Sheets, Level 2.

The attributes of LINE_COLOR are specified in Table 201

Table 201 – LINE_COLOR attribute

Usage	Attribute	Description
s	integer	The color to be used to display the user interface element.
s	reference	A reference to a VARIABLE instance containing the color to be used to display the user interface element.

7.36.11 LINE_TYPE

Purpose

The LINE_TYPE attribute specifies the type of data contained in a user interface element. Every user interface element with the same LINE_TYPE attribute should be displayed in the same fashion (i.e., line color, line pattern, line thickness, etc). By default, a user interface element without a LINE_TYPE attribute uses the DATA line type.

Lexical structure

LINE_TYPE ([DATA, n]*, LOW_LOW_LIMIT, LOW_LIMIT, HIGH_LIMIT, HIGH_HIGH_LIMIT, TRANSPARENT)<exp>

The attributes of LINE_TYPE are specified in Table 202.

Table 202 – LINE_TYPE attribute

Usage	Attribute	Description
s	DATA	The user interface element contains data.
c	n	An increasing integer starting with one, which identifies distinct DATA line types if more than one is present.
s	LOW_LOW_LIMIT	The user interface element contains a low low limit.
s	LOW_LIMIT	The user interface element contains a low limit.
s	HIGH_LIMIT	The user interface element contains a high limit.

Usage	Attribute	Description
s	HIGH_HIGH_LIMIT	The user interface element contains a high high limit.
s	TRANSPARENT	Only the user interface element data points are visible.

7.36.12 MEMBERS

Purpose

The MEMBERS attribute defines members of a basic construct. Each member specifies one item in the group, and is defined by a set of four attributes (identifier, reference, description, help).

The MEMBERS attribute of a FILE shall not be expressed using a conditional (IF, IF-ELSE, SELECT).

Lexical structure

MEMBERS [(identifier, reference, description, help)<exp>]+

The attributes of MEMBERS are specified in Table 203.

Table 203 – MEMBERS attributes

Usage	Attribute	Description
m	identifier	The name by which the item may be referenced. A specific identifier name can be reused in the MEMBERS attribute of multiple items as long as the items are of the same type.
m	reference	The reference to an EDD item. The type of the reference depends on the basic constructs. If MEMBERS is a CHART attribute, reference is a reference to a SOURCE instance. If MEMBERS is a COLLECTION attribute, reference is a reference to a AXIS, BLOCK_A, BLOCK_B, CHART, COLLECTION, COMMAND, CONNECTION, DOMAIN, EDIT_DISPLAY, FILE, GRAPH, LIST, MENU, METHOD, PROGRAM, RECORD, REFERENCE_ARRAY, REFRESH, RESPONSE_CODES, SOURCE, UNIT, VALUE_ARRAY, VARIABLE, VARIABLE_LIST, WAVEFORM, or WRITE_AS_ONE instance. If MEMBERS is a FILE or LIST attribute, reference is a reference to a COLLECTION, LIST, RECORD, REFERENCE_ARRAY, VALUE_ARRAY or VARIABLE instance. For a COLLECTION, the type of the COLLECTION shall be LIST, RECORD, REFERENCE_ARRAY, VALUE_ARRAY, or VARIABLE. If MEMBERS is a GRAPH attribute, reference is a reference to a WAVEFORM instance. If MEMBERS is a RECORDS attribute, reference is a reference to a VARIABLE instance. If MEMBERS is a SOURCE attribute, reference is a reference to a VARIABLE instance or a member of a RECORD instance. If MEMBERS is a VARIABLE_LISTS attribute, reference is a reference to a RECORDS, or VALUE_ARRAY or VARIABLE instance.
o	description	A short description of the item.
o	help	Help text for the item.

7.36.13 PROTOCOL

Purpose

The PROTOCOL attribute indicates communication protocol used to communicate to the device modeled by this EDD. The PROTOCOL is used to separate the EDDs within the catalog.

Lexical structure

PROTOCOL [PROFIBUS_DP, PROFIBUS_PA, PROFINET, FF, HART, string]

The attributes of PROTOCOL are specified in Table 204.

Table 204 – PROTOCOL attribute

Usage	Attribute	Description
s	FF	The component is a Fieldbus Foundation component.
s	HART	The component is a HART component.
s	PROFIBUS_DP	The component is a PROFIBUS DP component.
s	PROFIBUS_PA	The component is a PROFIBUS PA component.
s	PROFINET	The component is a PROFINET component.
s	string	Specifies any protocol. The protocol must be supported by the used EDD application.

7.36.14 RESPONSE_CODES**Purpose**

The RESPONSE_CODES attribute specifies values a device may return as error information. Each VARIABLE, RECORD, VALUE_ARRAY, VARIABLE_LIST, PROGRAM, COMMAND or DOMAIN can have its own set of RESPONSE_CODES.

Two lexical structures for the RESPONSE_CODES attribute are provided:

- the referenced RESPONSE_CODES contains a reference to a RESPONSE_CODES instance;
- the embedded RESPONSE_CODES allows to specify the RESPONSE_CODES elements directly at the element instance.

Lexical structure for referenced RESPONSE_CODES

RESPONSE_CODES (reference) <exp>

The attribute of the referenced RESPONSE_CODES is specified in Table 205.

Table 205 – RESPONSE_CODES attribute

Usage	Attribute	Description
m	reference	A reference to a RESPONSE_CODES instance.

Lexical structure for embedded RESPONSE_CODES

RESPONSE_CODES (description, integer, [DATA_ENTRY_ERROR, DATA_ENTRY_WARNING, MISC_ERROR, MISC_WARNING, MODE_ERROR, PROCESS_ERROR, SUCCESS], help) <exp>]+

The attributes of embedded RESPONSE_CODES are specified in Table 154.

7.36.15 SUPPLIED_INTERFACE**Purpose**

The SUPPLIED_INTERFACE attribute lists all interfaces this component supports.

Lexical structure

SUPPLIED_INTERFACE [interface-reference]+

The attributes of SUPPLIED_INTERFACE are specified in Table 206.

Table 206 – SUPPLIED_INTERFACE attribute

Usage	Attribute	Description
m	interface-reference	A list of interface identifiers.

7.36.16 VALIDITY

Purpose

The VALIDITY attribute specifies whether an element is valid or invalid. An element without a VALIDITY is always valid.

NOTE 1 VALIDITY should be expressed using a conditional (IF, IF-ELSE, SELECT).

NOTE 2 If a user interface element is invalid, it should not be displayed.

NOTE 3 A METHOD shall abort when an invalid VARIABLE is accessed.

Lexical structure

VALIDITY [(FALSE, TRUE)<exp>]

The attributes of VALIDITY are specified in Table 207.

Table 207 – VALIDITY attributes

Usage	Attribute	Description
s	FALSE	The element is invalid.
s	TRUE	The element is valid.

7.36.17 WIDTH

Purpose

The WIDTH attribute specifies the horizontal size of a user interface element when it is displayed. By default, the width of a user interface element without a WIDTH attribute is MEDIUM.

Lexical structure

WIDTH (XX_SMALL, X_SMALL, SMALL, MEDIUM, LARGE, X_LARGE, XX_LARGE)

The attributes of WIDTH are specified in Table 198.

7.37 Conditional expression

Purpose

Conditional expressions are used to declare alternative values or to calculate values for an attribute of a basic element. The value of the expression is evaluated during execution time. The IF or SELECT conditional can be used to enable or disable referenced instances of a list of ITEMS attribute of MENU, etc.

Three kinds of conditional expressions may be used.

- An IF conditional is used for specifying an attribute that has two alternative definitions. The expression specified in the IF conditional is evaluated. If the result is non-zero, the attribute is specified by a then-clause; otherwise, it is specified by an else-clause.
- A SELECT conditional is used for specifying an attribute that has several alternative definitions. The select-expression is evaluated and compared with every integer_n of CASE. If a match is found, the appropriate select-clause is valid. If no matching integer_n is found, the default-clause following DEFAULT is used.
- A primary, unary or binary expression (see 7.39.7). This kind of conditional expression can only be used for numeric values (e.g. DEFAULT_VALUE, MIN_VALUE, SLOT, INDEX).

EXAMPLE It is possible to define more than one value for a DEFAULT_VALUE, MIN_VALUE, etc. using IF- or SELECT-conditional.

Lexical structure for the IF conditional

IF (if-expression), (then-clause)+ ELSE (else-clause)+

The attributes are specified in Table 208.

Lexical structure for the SELECT conditional

SELECT select-expression, (CASE integer_n, select-clause)+, (DEFAULT, default-clause)

The attributes are specified in Table 208.

Table 208 – IF, SELECT conditional

Usage	Attribute	Description
m	if-expression	Evaluated to determine whether the then-clause or the else-clause is used to define an attribute. If the if-expression contains a reference to a VARIABLE instance, the value of the referenced VARIABLE is used.
m	then-clause	The definition for the attribute if the value of condition is non-zero. The clause structure depends on the attribute being defined. It can also take the form of another conditional.
o	else-clause	The definition for the attribute if the value of condition is zero. The clause structure depends on the attribute being defined. It can also take the form of another conditional.
m	select-expression	Evaluated to determine which select-clause is used to define an attribute. If the select-expression contains a reference to a VARIABLE instance, the value of the referenced VARIABLE is used.
m	integer_n	A number which identifies the according CASE. If integer matches with integer_n, the according select_clause is used.
m	select-clause	The definition for the attribute for each CASE, and the DEFAULT definition. The structure of each clause depends on the attribute being defined. Regardless of the attribute, each clause can also take the form of another conditional.
o	default-clause	The definition for the DEFAULT definition. The structure of default-clause depends on the attribute being defined. Regardless of the attribute, each clause can also take the form of another conditional.

If the

- then-clause,
- else-clause,
- select-clause,
- default-clause

contains references to METHOD instances or Builtins, the returned value of the referenced METHOD or Builtin shall be used within the clause. The return values shall be type-compatible.

7.38 Referencing

7.38.1 Referencing an EDD instance

Purpose

References are used within an EDD to refer to other EDD items.

EXAMPLE A PRE_EDIT_ACTIONS of VARIABLE refer to METHOD instances defined elsewhere in the EDD.

Lexical structure

The attribute is specified in Table 209.

Table 209 – Referencing an EDD instance

Usage	Attribute	Description
m	identifier	The identifier of an EDD instance.

7.38.2 Referencing bits of a BIT ENUMERATED VARIABLE

Purpose

This referencing is used to access a bit of a BIT ENUMERATED VARIABLE instance.

Lexical structure

variable-reference, bit-mask

The attributes are specified in Table 210.

Table 210 – Referencing elements of VARIABLE

Usage	Attribute	Description
m	variable-reference	A reference to a VARIABLE instance; the type of the VARIABLE instance shall be BIT ENUMERATED.
m	member-identifier	A bit-mask which identifies a single bit within the variable; this is the mask used to define the bit within the VARIABLE.

7.38.3 Referencing members of a RECORD

Purpose

This referencing is used to access a member of a RECORD instance.

Lexical structure

record-reference, member-identifier

The attributes are specified in Table 211.

Table 211 – Referencing elements of RECORD

Usage	Attribute	Description
m	record-reference	A reference to a RECORD instance.
m	member-identifier	The identifier of an item of MEMBERS.

7.38.4 Referencing elements of a VALUE_ARRAY

Purpose

This referencing is used to access an element of a VALUE_ARRAY instance.

Lexical structure

value-array-reference, expression

The attributes are specified in Table 183.

Table 212 – Referencing elements of VALUE_ARRAY

Usage	Attribute	Description
m	value-array-reference	A reference to a VALUE_ARRAY instance.
m	expression	An expression which shall be evaluated to a non-negative integer that is within the index range of the VALUE_ARRAY (for a description of expressions, see 7.40).

7.38.5 Referencing members of a COLLECTION

Purpose

This referencing is used to access a member of a COLLECTION instance.

Lexical structure

collection-reference, member-identifier

The attributes are specified in Table 213.

Table 213 – Referencing members of COLLECTION

Usage	Attribute	Description
m	collection-reference	A reference to a COLLECTION instance.
m	member-identifier	The identifier of an item of MEMBERS.

7.38.6 Referencing elements of a REFERENCE_ARRAY

Purpose

This referencing is used to access an element of a REFERENCE_ARRAY instance.

Lexical structure

reference-array-reference, expression

The attributes are specified in Table 214.

Table 214 – Referencing members of REFERENCE_ARRAY

Usage	Attribute	Description
m	reference-array-reference	A reference to a REFERENCE_ARRAY instance.
m	expression	An expression, which shall be evaluate to a non-negative integer that is within the index range of the REFERENCE_ARRAY (description of expressions see 7.39.7).

7.38.7 Referencing members of a VARIABLE_LISTS

Purpose

This referencing is used to access a member of a VARIABLE_LISTS instance.

Lexical structure

variable-list-reference, member-identifier

The attributes are specified in Table 215.

Table 215 – Referencing members of VARIABLE_LISTS

Usage	Attribute	Description
m	variable-list-reference	A reference to a VARIABLE_LISTS instance.
m	member-identifier	The identifier of an item of MEMBERS.

7.38.8 Referencing elements of BLOCK_A PARAMETERS

Purpose

This referencing is used to access an element of the PARAMETERS attribute of a BLOCK_A instance.

Lexical structure

PARAM, member-identifier

The attributes are specified in Table 216.

Table 216 – Referencing members of a BLOCK_A PARAMETERS

Usage	Attribute	Description
m	member-identifier	The identifier of an item of MEMBERS.

7.38.9 Referencing elements of BLOCK_A PARAMETER_LISTS

Purpose

This referencing is used to access an element of the PARAMETER_LISTS attribute of a BLOCK_A instance.

Lexical structure

PARAM_LIST, member-identifier

The attributes are specified in Table 217.

Table 217 – Referencing members of BLOCK_A PARAMETER_LISTS

Usage	Attribute	Description
m	member-identifier	The identifier of an item of MEMBERS.

7.38.10 Referencing elements of BLOCK_A LOCAL_PARAMETERS

Purpose

This referencing is used to access an element of the LOCAL_PARAMETERS attribute of a BLOCK_A instance.

Lexical structure

LOCAL_PARAM, member-identifier

The attributes are specified in Table 218.

Table 218 – Referencing members of BLOCK_A LOCAL_PARAMETER

Usage	Attribute	Description
m	member-identifier	The identifier of an item of MEMBERS.

7.38.11 Referencing BLOCK_A CHARACTERISTICS

Purpose

This referencing is used to access the CHARACTERISTICS of a BLOCK_A instance.

Lexical structure

BLOCK, characteristics-identifier

The attribute is specified in Table 219.

Table 219 – Referencing BLOCK_A CHARACTERISTICS

Usage	Attribute	Description
m	characteristics-identifier	The identifier of the CHARACTERISTICS reference.

7.38.12 Referencing members of a FILE

Purpose

This referencing is used to access a member of a FILE instance.

Lexical structure

file-reference, member-identifier

The attributes are specified in Table 220.

Table 220 – Referencing members of FILE

Usage	Attribute	Description
m	file-reference	A reference to a FILE instance.
m	member-identifier	The identifier of an item of MEMBERS.

7.38.13 Referencing elements of a LIST**Purpose**

This referencing is used to access an element of a LIST instance.

Lexical structure

list-reference, (expression, FIRST, LAST)

The attributes are specified in Table 221.

Table 221 – Referencing elements of LIST

Usage	Attribute	Description
m	list-reference	A reference to a LIST instance.
s	expression	An expression which shall be evaluate to a non-negative integer that is within the index range of the LIST (description of expressions see 7.39.7).
s	FIRST	The first element of the LIST.
s	LAST	The last element of the LIST.

7.38.14 Referencing members of a CHART**Purpose**

This referencing is used to access a member of a CHART instance.

Lexical structure

chart-reference, member-identifier

The attributes are specified in Table 222.

Table 222 – Referencing members of CHART

Usage	Attribute	Description
m	chart-reference	A reference to a CHART instance.
m	member-identifier	The identifier of an item of MEMBERS.

7.38.15 Referencing members of a GRAPH**Purpose**

This referencing is used to access a member of a GRAPH instance.

Lexical structure

graph-reference, member-identifier

The attributes are specified in Table 223.

Table 223 – Referencing members of GRAPH

Usage	Attribute	Description
m	graph-reference	A reference to a GRAPH instance.
m	member-identifier	The identifier of an item of MEMBERS.

7.38.16 Referencing members of a SOURCE

Purpose

This referencing is used to access a member of a GRAPH instance.

Lexical structure

source-reference, member-identifier

The attributes are specified in Table 224.

Table 224 – Referencing members of SOURCE

Usage	Attribute	Description
m	source-reference	A reference to a SOURCE instance.
m	member-identifier	The identifier of an item of MEMBERS.

7.38.17 Referencing AXIS of a GRAPH, SOURCE, WAVEFORM

Purpose

This referencing is used to access an AXIS of a GRAPH, SOURCE or WAVEFORM instance.

Lexical structure

reference, (X_AXIS, Y_AXIS)

The attributes are specified in Table 225.

Table 225 – Referencing AXIS of a GRAPH, SOURCE, WAVEFORM

Usage	Attribute	Description
m	reference	A reference to a GRAPH instance, a reference to a SOURCE instance, or a reference to a WAVEFORM instance.
c/o	X_AXIS	Only used with GRAPH; Y_AXIS used in all other cases.
c/o	Y_AXIS	Only used with SOURCE and WAVEFORM; X_AXIS used in all other cases.

7.38.18 Referencing PARAMETERS of specific BLOCK_A instance

Purpose

This referencing is used to access a member of the PARAMETERS attribute of a specific BLOCK_A instance.

Lexical structure

reference, expression, PARAM, member-identifier

The attributes are specified in Table 226.

Table 226 – Referencing PARAMETERS of specific BLOCK_A instance

Usage	Attribute	Description
m	reference	A reference to a BLOCK_A instance.
m	expression	A non-negative integer that specifies the occurrence of the BLOCK_A instance within the device.
m	member-identifier	The identifier of an item of MEMBERS.

7.38.19 Referencing LOCAL_PARAMETERS of specific BLOCK_A instance

Purpose

This referencing is used to access a member of the LOCAL_PARAMETERS attribute of a specific BLOCK_A instance.

Lexical structure

reference, expression, LOCAL_PARAM, member-identifier

The attributes are specified in Table 227.

Table 227 – Referencing LOCAL_PARAMETERS of specific BLOCK_A instance

Usage	Attribute	Description
m	reference	A reference to a BLOCK_A instance.
m	expression	A non-negative integer that specifies the occurrence of the BLOCK_A instance within the device.
m	member-identifier	The identifier of an item of MEMBERS.

7.38.20 Referencing CHARACTERISTICS of specific BLOCK_A instance**Purpose**

This referencing is used to access a member of the CHARACTERISTICS attribute of a specific BLOCK_A instance.

Lexical structure

reference, expression, BLOCK, member-identifier

The attributes are specified in Table 228.

Table 228 – Referencing CHARACTERISTICS of specific BLOCK_A instance

Usage	Attribute	Description
m	reference	A reference to a BLOCK_A instance.
m	expression	A non-negative integer that specifies the occurrence of the BLOCK_A instance within the device.
m	member-identifier	The identifier of an item of MEMBERS.

7.38.21 Referencing CHARTS of specific BLOCK_A instance**Purpose**

This referencing is used to access a member of the CHARTS attribute of a specific BLOCK_A instance.

Lexical structure

reference, expression, CHART, member-identifier

The attributes are specified in Table 229.

Table 229 – Referencing CHARTS of specific BLOCK_A instance

Usage	Attribute	Description
m	reference	A reference to a BLOCK_A instance.
m	expression	A non-negative integer that specifies the occurrence of the BLOCK_A instance within the device.
m	member-identifier	The identifier of an item of MEMBERS.

7.38.22 Referencing LISTS of specific BLOCK_A instance**Purpose**

This referencing is used to access a member of the LISTS attribute of a specific BLOCK_A instance.

Lexical structure

reference, expression, LIST, member-identifier

The attributes are specified in Table 230.

Table 230 – Referencing LISTS of specific BLOCK_A instance

Usage	Attribute	Description
m	reference	A reference to a BLOCK_A instance.
m	expression	A non-negative integer that specifies the occurrence of the BLOCK_A instance within the device.
m	member-identifier	The identifier of an item of MEMBERS.

7.38.23 Referencing GRAPHS of specific BLOCK_A instance

Purpose

This referencing is used to access a member of the GRAPHS attribute of a specific BLOCK_A instance.

Lexical structure

reference, expression, GRAPH, member-identifier

The attributes are specified in Table 231.

Table 231 – Referencing GRAPHS of specific BLOCK_A instance

Usage	Attribute	Description
m	reference	A reference to a BLOCK_A instance.
m	expression	A non-negative integer that specifies the occurrence of the BLOCK_A instance within the device.
m	member-identifier	The identifier of an item of MEMBERS.

7.38.24 Referencing GRIDS of specific BLOCK_A instance

Purpose

This referencing is used to access a member of the GRIDS attribute of a specific BLOCK_A instance.

Lexical structure

reference, expression, GRID, member-identifier

The attributes are specified in Table 232.

Table 232 – Referencing GRIDS of specific BLOCK_A instance

Usage	Attribute	Description
m	reference	A reference to a BLOCK_A instance.
m	expression	A non-negative integer that specifies the occurrence of the BLOCK_A instance within the device.
m	member-identifier	The identifier of an item of MEMBERS.

7.38.25 Referencing MENUS of specific BLOCK_A instance

Purpose

This referencing is used to access a member of the MENUS attribute of a specific BLOCK_A instance.

Lexical structure

reference, expression, MENU, member-identifier

The attributes are specified in Table 233.

Table 233 – Referencing MENUS of specific BLOCK_A instance

Usage	Attribute	Description
m	reference	A reference to a BLOCK_A instance.
m	expression	A non-negative integer that specifies the occurrence of the BLOCK_A instance within the device.
m	member-identifier	The identifier of an item of MEMBERS.

7.38.26 Referencing METHODS of specific BLOCK_A instance**Purpose**

This referencing is used to access a member of the METHODS attribute of a specific BLOCK_A instance.

Lexical structure

reference, expression, METHOD, member-identifier

The attributes are specified in Table 234.

Table 234 – Referencing METHODS of specific BLOCK_A instance

Usage	Attribute	Description
m	reference	A reference to a BLOCK_A instance.
m	expression	A non-negative integer that specifies the occurrence of the BLOCK_A instance within the device.
m	member-identifier	The identifier of an item of MEMBERS.

7.38.27 Referencing COMPONENT instances**Purpose**

This referencing is used to reference an instance that is related with a COMPONENT_RELATION from a component. If the reference is used within a MENU declaration and the instance does not exist, the referenced item will be treated as invalid. If the reference is used in a METHOD DECLARATION and the instance does not exist, the METHOD will be aborted. Such a reference is not allowed to use within a conditional expression.

Lexical structure

reference, expression, identifier

The attributes of component referencing are specified in Table 235.

Table 235 – Referencing a COMPONENT instance

Usage	Attribute	Description
o	reference	A reference to a COMPONENT.
o	expression	An expression, which shall be evaluate to a non-negative integer that is within the index range of the COMPONENT_RELATION.
o	identifier	The identifier of a basic construct of the referenced COMPONENT.

7.38.28 Referencing COMPONENT types

Purpose

This referencing is used to reference a component type. All conditional expressions are evaluated with INITIAL_VALUES or if not defined DEFAULT_VALUES or with data type defaults.

Lexical structure

reference, identifier

The attributes of component referencing are specified in Table 236.

Table 236 – Referencing a COMPONENT type

Usage	Attribute	Description
o	reference	A reference to a COMPONENT.
o	identifier	The identifier of a basic construct of the referenced COMPONENT.

7.39 Strings

7.39.1 Specifying a string as a string literal

Purpose

A text string can be specified as a string literal (see Annex A.2.3).

Lexical structure

string

The attribute is specified in Table 237.

Table 237 – string as a string literal

Usage	Attribute	Description
m	string	A string literal.

7.39.2 Specifying a string as a string variable

Purpose

A text string specified as a string variable is the value of the string variable.

Lexical structure

reference

The attributes is specified in Table 238.

Table 238 – string as a string variable

Usage	Attribute	Description
m	reference	A reference to a VARIABLE instance.

7.39.3 Specifying a string as an enumeration value

Purpose

An enumeration value string is a string associated with one of the values of an enumeration variable.

Lexical structure

reference, value

The attributes are specified in Table 239.

Table 239 – String as an enumeration value

Usage	Attribute	Description
m	reference	A reference to a VARIABLE instance of TYPE ENUMERATION.
m	value	The corresponding value of the description text within the enumeration list.

EXAMPLE The following enumeration value string specifies the string associated with the value 4 of the variable units_code: units_code (4)

7.39.4 Specifying a string as a dictionary reference

Purpose

A dictionary reference points to a string in a text dictionary.

Lexical structure

reference

The attribute is specified in Table 240.

Table 240 – String as a dictionary reference

Usage	Attribute	Description
m	reference	A reference to one or more strings specified in a dictionary (see 7.41).

7.39.5 Referencing HELP and LABEL attributes of EDD instances

Purpose

To get the current LABEL or the HELP string of an EDD instance, the instance and the LABEL or HELP attribute shall be referenced.

Lexical structure

reference, [HELP, LABEL]

The attributes are specified in Table 241.

Table 241 – Referencing HELP and LABEL attributes of EDD instances

Usage	Attribute	Description
m	reference	An EDD instance which contains the attributes LABEL and/or HELP.
s	HELP	The HELP string of the instance.
s	LABEL	The LABEL string of the instance.

7.39.6 String operations

Purpose

String literals, string variables, string enumeration values, dictionary references and attribute values of basic constructs can be concatenated with one another. If a term contains references of numeric data types, the values shall be converted to a string by the EDD application.

EXAMPLE variable_of_type_string = "Description"
 "Electronic " + "Device " + variable_of_type_string results in "Electronic Device Description".

Lexical structure

(string)<exp>, [(concatenation-operator, string)<exp>]+

The attributes are specified in Table 242.

Table 242 – String operation

Usage	Attribute	Description
m	string	Either a strings literals, string variables, string enumeration values, dictionary references or HELP, LABEL references.
o	concatenation-operator	The strings shall be concatenated.

7.39.7 Prompt string formats

Purpose

Prompt strings can be use to embed formatted items into strings. These items are composed of an optional format specifier and a VARIABLE ID or reference.

Lexical structure

format, (reference, id)

label, (reference, id)

The attributes are specified in Table 243.

Table 243 – Format specifier

Usage	Attribute	Description
m	format	Consists of the ANSI C (see ISO/IEC 9899) printf format string. If the format is not specified, the format of the referenced VARIABLE instance is used. If no format is specified, a default format is used based on the type of VARIABLE instance.
m	label	The LABEL of the VARIABLE instance.
o/s	reference	The value or label of the referenced VARIABLE instance.
o/s	id	The item IDs of the VARIABLES.

7.40 Expression

7.40.1 General structure

Purpose

There are three types of expressions.

- Primary expressions include constants, parenthesis-enclosed expressions, VARIABLE references, index element accesses or access to the status of VARIABLE instances. These operations are left-associative, i.e., they are evaluated from left to right.
- Unary expressions include all operators evaluating a single operand. These operations are right-associative, i.e., they are evaluated from right to left.
- Binary expressions include all operators evaluating two operands. These operations are left-associative, i.e., they are evaluated from left to right.

7.40.2 Primary expressions

Table 244 below summarizes the primary expressions in the EDDL.

Table 244 – Primary expressions

Primary expression	Description
constant	An expression whose value is identical to the value of the constant.
parenthesized expression	An expression whose value is identical to the value of the enclosed expression.
variable reference	An expression whose value is the value of the referenced VARIABLE.
ARRAY_INDEX	An expression whose value was taken from the index number of the referencing VALUE_ARRAY.

Primary expression	Description
attribute values of VARIABLES	A reference consisting of the identifier of a VARIABLE instance and an attribute. The reference supplies the according value of the attribute (see Table 245 for the returned values).
CURRENT_ROLE	A string constant which contains the current role. The value of CURRENT_ROLE shall be supported by the EDD application. The values for the string constant shall be defined by the ROLE attribute of MENU.
PI	The mathematical constant π .
attribute values of an AXIS	A reference to a AXIS instance and an attribute. The reference supplies the according value of the attributes (see Table 246 for the returned values). These attributes may only be read unless specified otherwise.
attribute values of a LIST	A reference to a LIST instance and an attribute. The reference supplies the according value of the attributes (see Table 247 for the returned values). These attributes may only be read unless specified otherwise.

Table 245 – Attribute values of VARIABLES

Attributes	Data type	Description
MIN_VALUE	type of VARIABLE instance	A value which complies with the MIN_VALUE of the VARIABLE instance. If more than one MIN_VALUE is defined, MIN_VALUE plus the range identifier (see Table 165) shall be used for referencing.
MAX_VALUE	type of VARIABLE instance	A value which complies with the MAX_VALUE of the VARIABLE instance. If more than one MAX_VALUE is defined, MAX_VALUE plus the range identifier (see Table 165) shall be used for referencing.
SCALING_FACTOR	double	A value which complies with the SCALING_FACTOR of the VARIABLE instance.
DEFAULT_VALUE	type of VARIABLE instance	A value which complies with the DEFAULT_VALUE of the VARIABLE instance.
INITIAL_VALUE	type of VARIABLE instance	A value which complies with the INITIAL_VALUE of the VARIABLE instance.
VARIABLE_STATUS		A value which indicates the status of VARIABLE. The following status are defined: VARIABLE_STATUS_NONE specifies that the value is valid and not one of the other status. VARIABLE_STATUS_INVALID specifies that the value is out of range. VARIABLE_STATUS_NOT_ACCEPTED specifies that the value could not transferred. VARIABLE_STATUS_NOT_SUPPORTED specifies that the value is not supported. VARIABLE_STATUS_CHANGED specifies that the value has been changed by the user or by a method. VARIABLE_STATUS_LOADED specifies that the value has been successful loaded. VARIABLE_STATUS_INITIAL specifies that the value is unchanged. VARIABLE_STATUS_NOT_CONVERTED specifies that the value has not been converted, for example, because the units are not compatible.

Table 246 – AXIS Attribute Values

Attributes	Data type	Description
MIN_VALUE	arithmetic type	A value which complies with the MIN_VALUE of the AXIS instance.
MAX_VALUE	arithmetic type	A value which complies with the MAX_VALUE of the AXIS instance.
SCALING	LINEAR or LOGARITHMIC	A value which complies with the SCALING of the AXIS

Attributes	Data type	Description
		instance.
VIEW_MIN	arithmetic type	A value which complies with the VIEW_MIN of the AXIS instance; VIEW_MIN can be read and written.
VIEW_MAX	arithmetic type	A value which complies with the VIEW_MAX of the AXIS instance; VIEW_MAX can be read and written.

Table 247 – LIST Attribute Values

Attributes	Data type	Description
CAPACITY	unsigned_integer	A value which complies with the CAPACITY of the LIST instance.
COUNT	unsigned_integer	A value which complies with the COUNT of the LIST instance.

7.40.3 Unary expressions

A unary expression consists of a numerical operand, an expression, preceded by a unary operator. Table 248 specifies the unary expressions in the EDDL.

Table 248 – Unary expressions

Operator	Description
-	The arithmetic negation of its operand.
~	The bitwise negation of its operand, that is, each bit of the result is the inverse of the corresponding bit of the operand. The operand of the ~ operator shall have an integral value.
!	The logical negation of its operand.

7.40.4 Binary expressions

7.40.4.1 General structure

A binary expression consists of two operands or expressions, separated by a binary operator. If either operand has a floating-point value, the other operand is converted (promoted) to a floating-point value. This subsection specifies the following types of binary expressions.

- Multiplicative
- Additive
- Shift
- Relational
- Equality
- Bitwise AND (&)
- Bitwise XOR (^)
- Bitwise OR (|)
- Logical AND (&&)
- Logical OR (||)
- Conditional evaluation

7.40.4.2 Multiplicative operators

Multiplicative operators specify multiplication and division of operands. Table 249 specifies the multiplicative operators.

Table 249 – Multiplicative operators

Operator	Description
*	The multiplication of its operands.

Operator	Description
/	The division of the first operand by the second operand. The result of the / operator is the quotient of the division.
%	The division of the first operand by the second operand. The result of the % operator is the remainder.

7.40.4.3 Additive operators

Additive operators specify the addition and subtraction of numerical operands. If this operator is applied on numerical operands, then the result is the sum of its two operands, and the operation is commutative. If the + operator is applied on string operands, then the result is a string value generated by appending the second string to the first string, and the operation is not commutative.

The minus operator indicates subtraction. The resulting value of this operation is the difference of its operands. The second operand is subtracted from the first. Table 250 specifies additive operators.

Table 250 – Additive operators

Operator	Description
+	The addition of its operands.
-	The subtraction of the second operand from the first.

7.40.4.4 Shift operators

The << and >> operators specify a shift of the first operand by the number of bits specified by the second operand. The operands of the << and >> operators shall have integer values. Table 251 specifies the shift operators.

Table 251 – Shift operators

Operator	Description
<<	The first operand to the left. The bits shifted off are discarded, and the vacated bits are zero filled.
>>	The first operand to the right. The bits shifted off are discarded. If the first operand is less than 0, the vacated bits are one filled; otherwise they are zero filled.

7.40.4.5 Relational operators

Relational operators (<, <=, >, >=) specify a comparison of its operands. The result of this type of expression is 1 if the tested relationship is true; otherwise, the result is 0. The operands shall be numerical data types. Table 252 specifies the relational operators.

Table 252 – Relational operators

Operator	Description
<	Tests for the relationship "less than".
<=	Tests for the relationship "less than or equal to".
>	Tests for the relationship "greater than".
>=	Tests for the relationship "greater than or equal to".

7.40.4.6 Equality operators

The equality operators are == and !=. The result of this type of expression is 1 if the tested relationship is true; otherwise, the result is 0. The operators shall be of the same data type. Any data types are allowed. Table 253 specifies the equality operators.

Table 253 – Equality operators

Operator	Description
==	Tests for the relationship “equals”.
!=	Tests for the relationship “does not equal”.

7.40.4.7 Bitwise AND operator (&)

The & operator specifies the bitwise AND of its integer operands, that is, each bit of the result is set if each of the corresponding bits of the operands is set. The operands of the & operator shall have integral values.

7.40.4.8 Bitwise XOR operator (^)

The ^ operator specifies the bitwise exclusive OR of its integer operands, that is, each bit of the result is set if only one of the corresponding bits of the operands is set. The operands of the ^ operator shall have integral values.

7.40.4.9 Bitwise OR operator (|)

The | operator specifies the bitwise inclusive OR of its integer operands, that is, each bit of the result is set if either of the corresponding bits of the operands is set. The operands of the | operator shall have integral values.

7.40.4.10 Logical AND operator (&&)

The && operator specifies the Boolean AND evaluation of its integer operands. If either operand references an invalid VARIABLE (VALIDITY equal to FALSE), the operand shall evaluate to 0. The result of this type of expression is 1 if both of the operands are not equal to 0; otherwise, the result is 0. If the first operand is equal to 0, the second operand is not evaluated.

7.40.4.11 Logical OR operator (||)

The || operator specifies the Boolean OR evaluation of its integer operands. If either operand references an invalid VARIABLE (VALIDITY equal to FALSE), the operand shall evaluate to 0. The result of this type of expression is 1 if either of the operands is not equal to 0; otherwise, the result is 0. If the first operand is not equal to 0, the second operand is not evaluated.

7.40.4.12 Conditional evaluation

The operators ? and : are used for conditional evaluation.

EXAMPLE 1

```
result = logexpr ? trueexpr : falseexpr;
logexpr is evaluated and if it is non-zero, the result is the value of the trueexpr. Otherwise, the falseexpr is executed.
```

EXAMPLE 2 The following if-else structure is logical identical to example 1:

```
if (logexpr)
    result = trueexpr ;
else
    result = falseexpr ;
```

7.41 Text dictionary**General structure**

The dictionary is a collection of text strings that can be used in an EDD. The dictionary text is referenced within an EDD. The dictionary can be accessed from any EDD to assure consistency of text strings (for example, used for LABEL and HELP attributes).

Generally, one common text dictionary is used for a group of device types. Additionally, a user specific dictionary can be specified. The text dictionary supports multilingual texts.

Lexical structure

```
identifier (language-code, country-code, string)+
```

The attributes are specified in Table 254.

Table 254 – Text dictionary attributes

Usage	Attribute	Description
m	identifier	The name of a text string in an EDD.
m	language-code	The language used and the ASCII or multi code presentation of a string; language codes are defined in ISO 639.
o	country-code	The country in which the language is spoken; country codes are defined in ISO 3166-1.
m	string	The text string in the related language.

In addition to the standard country codes defined in ISO 3166-1, the ZZ country code may be used to specify a short version of a string. Short versions of strings are useful for EDD applications that run on mobile devices that have a limited screen size.

EXAMPLE

```

write_protected
"Write Protected"
"|de|Schreibgeschützt"
"|fr|Protégé en écriture"
"|it|Protetto in scrittura"
"|es|Protegido contra escritura"
"|en ZZ|Wr Prot"
"|de ZZ|Geschützt"

```

The identifier "write_protected" is the identifier which is referenced within the EDD. The qualifiers |de|, |fr|, |it|, |es| specify the text for the languages German, French, Italian and Spain. The "en ZZ" and "de ZZ" qualifiers specify short versions of the English and German strings. This same approach could also be used to provide short versions for French, Italian, and Spanish.

Annex A (normative)

EDDL formal definition

A.1 EDDL preprocessor

A.1.1 General structure

Purpose

The preprocessor generates an EDD before compilation. The preprocessor replaces parts of the EDD text, inserts the contents of other files or suppresses the processing of parts of the EDD by removing sections.

Structure

All preprocessor directives start with a hash symbol (#) as the first character on a line. A space (SPACE or TAB characters) can appear after the initial # for proper indentation.

A.1.2 Directives

A.1.2.1 #define

Purpose

The #define directive gives a meaningful name to a constant in the EDD.

Structure

```
#define identifier token-string
```

This form of the #define directive instructs the preprocessor to replace all subsequent occurrences of the identifier in the EDD file with token-string.

NOTE The identifier is not replaced if it appears in a comment, within a string, or as part of a longer identifier.

```
#define identifier
```

This form of the #define directive instructs the preprocessor to remove all occurrences of identifier from the EDD file. The identifier remains defined and can be tested using the #if defined and #ifdef directives.

```
#define identifier(argument, ... ,argument) token-string
```

This form of the #define directive instructs the preprocessor to create function-like macros. This form accepts a list of arguments that shall appear in parentheses and are separated by commas.

When a macro has been defined in this syntax form, subsequent textual instances followed by an argument list constitute a macro call. The actual arguments following an instance of identifier in the source file are matched to the corresponding formal parameters in the macro definition. Each formal parameter in token-string is replaced by the corresponding actual argument. Any macros in the actual argument are expanded before the directive replaces the formal parameter.

Each argument in the list shall be unique.

No spaces are allowed to separate identifier and the opening parenthesis.

For long directives on multiple source lines, a backslash (\) shall be used before the NEWLINE character.

No spaces are allowed between name and the “(“.

A.1.2.2 #include**Purpose**

The `#include` directive tells the preprocessor to treat the contents of a specified file as if those contents had appeared in the source program at the point where the directive appears.

Structure

```
#include "filename"
#include <filename>
```

Read the contents of the file named with filename. The preprocessor processes this data as if it was part of the current file.

Both syntax forms cause replacement of that directive by the entire contents of the specified include file. The difference between the two forms is the order in which the preprocessor searches for include files when the path is incompletely specified.

No additional tokens are permitted on the directive line after the final `"` or `>`.

A.1.2.3 #line**Purpose**

The `#line` directive causes the preprocessor to change the compiler's internally stored line number and filename to a given line number and filename. The EDD processor uses the line number and filename to refer to errors that it finds during processing. The line number usually refers to the current input line, and the filename refers to the current input file. The line number is incremented after each line is processed.

Structure

```
#line integer-constant "filename"
```

integer-constant and filename causes the preprocessor to substitute the internally stored line number of the EDD processor to a given line number and filename. Integer-constant is interpreted as the line number of the next line and is incremented after each line is processed.

A.1.2.4 #if, #elif, #else, and #endif**Purpose**

The `#if` directive, with the `#elif`, `#else`, and `#endif` directives, controls processing parts of a EDD file.

```
#if constant-expression
```

Subsequent lines up to the `#else`, `#elif` or `#endif` directive, appear in the output only if constant-expression yields a non-zero value. The binary operators `"&&"` and `"||"` are legal in constant-expression as well as the `"?:"` operator and the unary `"-"`, `"!"` and `"~"` operators.

In addition the unary operator defined can be used in special constant expressions, as shown by the following syntax:

```
defined(identifier)
defined identifier
```

This constant expression is considered true (non-zero) if the identifier is currently defined; otherwise, the condition is false. An identifier defined as empty text is considered defined. The `defined` directive can be used in an `#if` and an `#elif` directive, but nowhere else.

```
#elif constant-expression
```

Any number of `#elif` directives may appear between an `#if`, `#ifdef` or `#ifndef` directive and a matching `#else` or `#endif` directive. The lines following the `#elif` directive appear in the output only if all the following conditions hold.

- The constant-expression in the preceding `#if` directives is evaluated to zero, the name in the preceding `#ifdef` is not defined or the name in the preceding `#ifndef` directives was defined.
- The constant-expressions in all inverting `#elif` directives are evaluated to zero.
- The current constant-expression evaluates to non-zero.

If the constant-expression evaluates to non-zero, ignore subsequent `#elif` and `#else` directives up to the matching `#endif`. Any constant-expression allowed in an `#if` directive is allowed in an `#elif` directive.

`#else`

If all occurrences of constant-expression are false, or if no `#elif` directives appear, the preprocessor selects the text block after the `#else` clause. If the `#else` clause is omitted and all instances of constant-expression in the `#if` block are false, no text block is selected.

`#endif`

Ends a section of lines which starts with one of the conditionals directives `#if`, `#ifdef` or `#ifndef`. Each directive shall have a matching `#endif`.

NOTE 1 The preprocessor recognizes formal parameters for macros in `#define` directive bodies, even when they occur outside character constants and quoted strings. Macro names are not recognized within character constants or quoted strings during the regular scan. Thus `#define aaa bbb` does not expand `aaa` in LABEL "aaa".

NOTE 2 Macros are not expanded while processing a `#define` or `#undef`. Thus:

```
#define aaa bbb
#define ccc aaa
#undef aaa
ccc
produces aaa.
```

NOTE 3 Macros are not expanded during the scan which determines the actual parameters to another macro call.

```
Thus:
#define turn(aaa,bbb) bbb aaa
#define ccc ddd
turn(ccc, #define ccc eee)
produces #define ddd eee ddd.
```

A.1.2.5 #ifdef, #ifndef and #undef

Purpose

The `#ifdef` and `#ifndef` directives control processing parts of an EDD file. The `#ifdef` and `#ifndef` directives perform the same task as the `#if` directive when it is used with defined identifier.

`#ifdef identifier`

Subsequent lines up to the matching `#else`, `#elif` or `#endif` appear in the output only if the identifier has been defined. The identifier is specified using either a `#define` directive or outside the EDD and in absence of an intervening `#undef` directive. No additional tokens are permitted on the directive line after name.

`#ifndef identifier`

Subsequent lines up to the matching `#else`, `#elif` or `#endif` appear in the output only if name has not been defined or if its definition has been removed with an `#undef` directive. No additional tokens are permitted on the directive line after name.

`#undef identifier`

The `#undef` directive causes an identifier's preprocessor definition to be removed. No additional tokens are permitted on the directive line after name.

A.1.3 Predefined macros

A.1.3.1 General structure

The preprocessor shall recognize a specified set of predefined macros. These macros should be used anywhere in the EDD and are for informational purpose.

These macros take no arguments and cannot be redefined.

A.1.3.2 List of predefined macros

A.1.3.2.1 `__FILE__`

Purpose

`__FILE__` contains the name of the current EDD file.

Structure

`__FILE__`

`__FILE__` expands to a string surrounded by double quotation marks.

A.1.3.2.2 `__LINE__`

Purpose

`__LINE__` contains the line number in the current EDD file.

Structure

`__LINE__`

`__LINE__` is a decimal integer constant. It can be altered with a `#line` directive.

A.1.4 NEWLINE characters

A NEWLINE character terminates a character constant or a quoted string. The backslash (\) followed by a NEWLINE in the body of a `#define` statement is used to continue the definition onto the next line.

A.1.5 Comments

A comment is a sequence of characters beginning with a forward slash/asterisk combination (`/*`) and ends with asterisk/slash combination (`*/`). A comment can include any combination of characters from the representable character set, including NEWLINE characters.

Furthermore, the single-line comment is supported if it is preceded by two forward slashes (`//`). A comment beginning with two forward slashes is terminated by the next NEWLINE character that is not preceded by an escape character.

A.2 Conventions

A.2.1 Integer constants

An integer constant may be specified in binary, octal, decimal, or hexadecimal notation. Table A.1 specifies the conventions for each type of notation.

Table A.1 – Conventions for integer constants

Integer Constant Type	Conventions
binary	A non-empty sequence of the binary digits 0 and 1 preceded by either 0b or 0B.
octal	A non-empty sequence of the digits 0 through 7 beginning with 0.
decimal	A non-empty sequence of the decimal digits 0 through 9, not beginning with 0.
hexadecimal	A non-empty sequence of hexadecimal digits preceded by either 0x or 0X. The hexadecimal digits are the digits 0 through 9 and the letters a through f (or A through F) with the values 10 through 15, respectively.

A.2.2 Floating-point constants

Lexical structure

A floating-point constant has four parts:

- an integer part, a sequence of decimal digits;
- a decimal point (.);
- a fraction part, a sequence of decimal digits;
- an exponent part, a possibly signed sequence of decimal digits preceded by one of the letters e or E.

Rules

The following rules apply to using floating-point constants:

- either the integer part or the fraction part can be omitted, but not both;
- either the decimal point or the exponent part can be omitted, but not both.

Example: Following are examples of floating point constants:

59,
,87
48,93
4,8e12

The calculation of the algebraic expression shall use the most precise data type.

A.2.3 String literals

String literals in an EDD shall be encoded using either ISO Latin-1 (ISO/IEC 8859-1) or UTF-8 (RFC 3629). Within a given EDD all string literals shall use the same encoding. A string literal is a possibly empty sequence of characters enclosed in double quotes ("). The enclosed characters shall be any character except the following:

- double quote (");
- backslash (\);
- new line.

Using escape sequences in string literals a string constant can also contain escape sequences that represent a character. Table A.2 shows the escape sequences and their result.

Table A.2 – Using escape sequences in string literals

Escape code	Result
'	Single quote.
"	Double quote.
	Vertical bar.
?	Question mark.
\	Backslash.
\a	Alert.
\f	Form feed.
\n	New line.
\r	Carriage return.
\t	Horizontal tab.
\v	Vertical tab.

A.2.4 Using language codes in string constants

A string literal can encapsulate all the translations of a given phrase. The language code is specified using ISO/IEC 8859-1. If a string literal does not contain translations for all the languages, a default language shall be used. Table A.3 shows examples of language codes.

Table A.3 – Using language codes in string literals

Language Code	Language
no code assigned	Default language, used if a language is requested that is not defined.
zh	Chinese
de	German
en	English
es	Spanish
fr	French
it	Italian
ja	Japanese
ko	Korean

NOTE English is recommended as the default language.

EXAMPLE The following string literal specifies the English phrase "Invalid Selection" in English and German:

"Invalid Selection"

"|de|Unzulässige Auswahl"

A.3 Operators

Table A.4 contains operators of the EDDL.

Table A.4 – EDDL operators

!	=	%	%=
&	&&	&=	(
)	*	*=	+
++	+=	,	-
--	-=	.	/
/=	:	;	<
<<	<<=	<=	=
=	>	>=	>>
>>=	?	[	]
^	^=	{	
=		}	~
->			

A.4 Keywords

Table A.5 contains the EDDL keywords.

Table A.5 – EDDL keywords

EDDL keyword	EDDL keyword	EDDL keyword	EDDL keyword
ACCESS	DOMAINS	MAX_VALUE6	SCAN_LIST
ACTUATOR	double	MAX_VALUE7	SCOPE
ACTUATOR_ELECTRIC	DOUBLE	MAX_VALUE8	SECONDS
ACTUATOR_ELECTRO_PNEUMATIC	DURATION	MAX_VALUE9	SELECT
ACTUATOR_HYDRAULIC	DYNAMIC	MIN_VALUE	SELF
ADD	EDD	MIN_VALUE1	SELF_CORRECTING
ADDRESSING	EDD_PROFILE	MIN_VALUE2	SEMICONDUCTOR
ALARM	EDD_REVISION	MIN_VALUE3	SENSOR
ANALOG_INPUT	EDD_VERSION	MIN_VALUE4	SENSOR_ACOUSTIC
ANALOG_OUTPUT	EDIT_DISPLAY	MIN_VALUE5	SENSOR_ANALYTIC
APPINSTANCE	EDIT_DISPLAY_ITEMS	MIN_VALUE6	SENSOR_ANALYTIC_GAS
ARGUMENTS	EDIT_DISPLAYS	MIN_VALUE7	SENSOR_ANALYTIC_LIQUID
ARRAY	EDIT_FORMAT	MIN_VALUE8	SENSOR_CONCENTRATION
ARRAY_INDEX	EDIT_ITEMS	MIN_VALUE9	SENSOR_DENSITY
ARRAYS	ELECTRICAL_DISTRIBUTION	MINIMUM_NUMBER	SENSOR_FLOW
ASCII	ELECTRICAL_DISTRIBUTION_POWER_MONITORING	MINIMUM_NUMBER	SENSOR_FLOW_CORIOLIS
AUTO_CREATE	ELEMENTS	MINUTES	SENSOR_FLOW_ELECTROMAGNETIC
AXES	else	MINUTES	SENSOR_FLOW_MECHANICAL
AXIS	ELSE	MISC	SENSOR_FLOW_RADIOMETRIC
AXIS_ITEMS	EMPHASIS	MISC	SENSOR_FLOW_THERMAL
BAD	ENUMERATED	MISC_ERROR	SENSOR_FLOW_ULTRASONIC
BIT_ENUMERATED	EUC	MISC_ERROR	SENSOR_FLOW_VORTEX_COUNTER
BITSTRING	EVENT	MISC_WARNING	SENSOR_LEVEL
BLOCK	EVERYTHING	MISC_WARNING	SENSOR_LEVEL_BUOYANCY
BLOCKS	EXIT_ACTIONS	MODE	SENSOR_LEVEL_CAPACITIVE
break	FALSE	MODE	SENSOR_LEVEL_ECHO
CAN_DELETE	FF	MODE_ERROR	SENSOR_LEVEL_HYDROSTATIC
CAPACITY	FILE	MODE_ERROR	SENSOR_LEVEL_RADIOMETRIC
case	FILE_ITEMS	MODULE	SENSOR_MULTIVARIABLE
CASE	FILES	MODULE	SENSOR_POSITION
CATALOG	FILTER	MORE	SENSOR_PRESSURE
char	FIRST	MORE	SENSOR_TEMPERATURE
CHARACTERISTICS	float	NETWORK_COMPONENT	SEPARATOR
CHART	FLOAT	NETWORK_COMPONENT	SERVICE
CHART_ITEMS	for	NETWORK_COMPONENT_WIRELESSADAPTER	short
CHARTS	FREQUENCY_CONVERTER	NEXT	SIBLING_COMPONENT
CHECK_CONFIGURATION	FREQUENCY_INPUT	NEXT	signed
CHILD	FREQUENCY_OUTPUT	NEXT_COMPONENT	SIMULATION
CHILD_COMPONENT	FUNCTION	NEXT_COMPONENT	SLOT
CLASS	GAUGE	NO_LABEL	SMALL
CLASSIFICATION	GRAPH	NO_UNIT	SOFTWARE
COLLECTION	GRAPH_ITEMS	NUMBER	SOURCE
COLLECTION_ITEMS	GRAPHS	NUMBER_OF_ELEMENTS	SOURCE_ITEMS

EDDL keyword	EDDL keyword	EDDL keyword	EDDL keyword
COLLECTIONS	GRID	NUMBER_OF_POINTS	SOURCES
COLUMNBREAK	GRID_ITEMS	OBJECT_REFERENCE	STATE
COMM	GRIDS	OCTET	STRIP
COMMAND	GROUP	OF	STYLE
COMMANDS	HANDLING	OFFLINE	SUB_SLOT
COMPONENT	HARDWARE	ONLINE	SUCCESS
COMPONENT_FOLDER	HART	OPERATE	SUMMARY
COMPONENT_FOLDERS	HEADER	OPERATION	SUPPLIED_INTERFACE
COMPONENT_PARENT	HEIGHT	OPTIONAL	SWEEP
COMPONENT_PATH	HELP	ORIENTATION	switch
COMPONENT_REDEFINITIONS	HIDDEN	OUTPUT	SWITCHGEAR
COMPONENT_REFERENCE	HIGH_HIGH_LIMIT	PACKED_ASCII	TABLE
COMPONENT_REFERENCES	HIGH_LIMIT	PAGE	TEMPLATE
COMPONENT_RELATION	HORIZONTAL	PARAM	TEMPLATES
COMPONENT_RELATIONS	HORIZONTAL_BAR	PARAM_LIST	TEMPORARY
COMPONENTS	HOURS	PARAMETER_LISTS	TIME
COMPUTATION	if	PARAMETERS	TIME_FORMAT
CONNECTION	IF	PARENT	TIME_SCALE
CONNECTIONS	IGNORE_IN_HOST	PARENT_COMPONENT	time_t
CONSTANT_UNIT	IGNORE_IN_TEMPORARY_MASTER	PASSWORD	TIME_VALUE
CONTAINED	IMAGE	PATH	TRANSACTION
continue	IMAGE_ITEMS	PHYSICAL	TRANSDUCER
CONTROLLER	IMAGES	PI	TRANSPARENT
CONVERTER	IMPORT	POST_EDIT_ACTIONS	TRUE
CORRECTABLE	INDEX	POST_READ_ACTIONS	TUNE
CORRECTION	INDICATOR	POST_WRITE_ACTIONS	TYPE
COUNT	INFO	PRE_EDIT_ACTIONS	UNCORRECTABLE
CURRENT_ROLE	INIT_ACTIONS	PRE_READ_ACTIONS	UNIT
CYCLE_TIME	INITIAL_VALUE	PRE_WRITE_ACTIONS	UNIT_ITEMS
DATA	INITIAL_VALUES	PREV	UNITS
DATA_ENTRY_ERROR	INLINE	PREV_COMPONENT	UNIVERSAL
DATA_ENTRY_WARNING	INPUT	PROCESS	unsigned
DATA_EXCHANGE	int	PROCESS_ERROR	UNSIGNED_INTEGER
DATA1	INTEGER	PROCESS_VALUE	VALIDITY
DATA2	INTERFACE	PROFIBUS_DP	VARIABLE
DATA3	INTERFACES	PROFIBUS_PA	VARIABLE_LIST
DATA4	ITEM_ARRAY	PROFINET	VARIABLE_LISTS
DATA5	ITEM_ARRAY_ITEMS	PROGRAM	VARIABLE_STATUS
DATA6	ITEM_ARRAYS	PROGRAMS	VARIABLES
DATA7	ITEMS	PROTOCOL	VECTORS
DATA8	KEY_POINTS	PURPOSE	VERTICAL
DATA9	LABEL	RDEFINE	VERTICAL_BAR
DATE	LARGE	READ	VIEW_MAX
DATE_AND_TIME	LAST	READ_ONLY	VIEW_MIN
DD_ITEM	LENGTH	READ_TIMEOUT	VISIBLE
DD_REVISION	LIKE	RECORD	VISIBLE
DD_STRING	LINE_COLOR	RECORDER	WARNING
DECLARATION	LINE_TYPE	RECORDS	WAVEFORM
default	LINEAR	REDEFINE	WAVEFORM_ITEMS

EDDL keyword	EDDL keyword	EDDL keyword	EDDL keyword
DEFAULT	LINK	REDEFINITIONS	WAVEFORMS
DEFAULT_VALUE	LIST	REDUNDANCY	while
DEFAULT_VALUES	LIST_ITEMS	REFRESH	WIDTH
DEFINITION	LISTS	REFRESH_ACTIONS	WINDOW
DELETE	LOAD_TO_APPLICATION	REFRESH_ITEMS	WRITE
DETAIL	LOAD_TO_DEVICE	REFRESHES	WRITE_AS_ONE
DETECT	LOCAL	RELATION_TYPE	WRITE_AS_ONE_ITEMS
DEVICE	LOCAL_DISPLAY	RELATIONS	WRITE_AS_ONES
DEVICE_REVISION	LOCAL_PARAM	REMOTEIO	WRITE_TIMEOUT
DEVICE_TYPE	LOCAL_PARAMETERS	REPLY	X_AXIS
DIAGNOSE	LOGARITHMIC	REQUEST	X_INCREMENT
DIAGNOSTIC	long	REQUIRED_INTERFACE	X_INITIAL
DIALOG	LOW_LIMIT	REQUIRED_RANGES	X_LARGE
DIGITAL_INPUT	LOW_LOW_LIMIT	RESPONSE_CODES	X_SMALL
DIGITAL_OUTPUT	MANUFACTURER	return	X_VALUES
DISCRETE_IN	MANUFACTURER_EXT	REVIEW	XX_LARGE
DISCRETE_INPUT	MAX_VALUE	ROLE	XX_SMALL
DISCRETE_OUT	MAX_VALUE1	ROOT	XY
DISCRETE_OUTPUT	MAX_VALUE2	ROWBREAK	Y_AXIS
DISPLAY_FORMAT	MAX_VALUE3	SCALING	Y_VALUES
DISPLAY_ITEMS	MAX_VALUE4	SCALING_FACTOR	YT
DOMAIN	MAX_VALUE5	SCAN	

A.5 Terminals

The following text contain allowed terminals of the EDDL lexical structure.

```

DEFINE digit          = { 0-9 } .
    bin_digit         = { 0 1 } .
    non_zero_digit    = { 1-9 '-' } .
    oct_digit         = { 0-7 } .
    hex_digit         = { 0-9abcdefABCDEF } .
    letter            = { a-zA-Z } .
    escapes           = { '\"?afnrvt'\\" } .
    ISOLatin1char     = - { \" } .

/* Integer */
(0b|0B) bin_digit +    /* binary */
non_zero_digit digit * /* decimal */
"0" oct_digit *        /* octal */
(0x|0X) hex_digit +    /* hexadecimal */

/* Real */
digit*"."digit+((E|e){+\\-}?digit+)?

/* String */
\" ISOLatin1char * \"
\" UTF8char * \"

/* Character */
\' ISOLatin1char \'
\' UTF8char \'

```

NOTE An EDD shall use one and only one character encoding scheme (i.e., ISO Latin-1 or UTF-8)

```
/* Identifier */
```

letter (letter|digit|_)*

A.6 Formal EDDL syntax

A.6.1 General

This annex contains a formal syntax of the EDDL using the Backus Naur Form.

If there are optional elements then each element is marked with /* M */ if the element is mandatory or /* O */ if the element is optional.

Each Backus Naur Form rule is identified by an identifier followed by a colon.

A.6.2 EDD identification information

```
device_description:
    = identification definition_list
```

```
identification:
    = manufacturer ',' device_type ',' device_revision ','
      DD_revision
    = manufacturer ',' device_type ',' device_revision ','
      DD_revision ',' edd_version
    = manufacturer ',' device_type ',' device_revision ','
      DD_revision ',' edd_profile
    = manufacturer ',' device_type ',' device_revision ','
      DD_revision ',' manufacturer_ext
    = manufacturer ',' device_type ',' device_revision ','
      DD_revision ',' edd_version ',' edd_profile
    = manufacturer ',' device_type ',' device_revision ','
      DD_revision ',' edd_version ',' manufacturer_ext
    = manufacturer ',' device_type ',' device_revision ','
      DD_revision ',' edd_profile ',' manufacturer_ext
    = manufacturer ',' device_type ',' device_revision ','
      DD_revision ',' edd_version ',' edd_profile ','
      manufacturer_ext
```

```
manufacturer:
    = 'MANUFACTURER' Integer
    = 'MANUFACTURER' Identifier
```

```
device_type:
    = 'DEVICE_TYPE' Integer
    = 'DEVICE_TYPE' Identifier
```

```
device_revision:
    = 'DEVICE_REVISION' Integer
```

```
DD_revision:
    = 'DD_REVISION' Integer
    = 'EDD_REVISION' Integer
```

```
edd_version:
    = 'EDD_VERSION' Integer
```

```
edd_profile:
    = 'EDD_PROFILE' Integer
```

```
manufacturer_ext:
    = 'MANUFACTURER_EXT' string
```

```
definition_list:
    = definition
```

```
= definition_list definition
```

```
definition:
```

```
= item
= imported_description
= like
```

```
item:
```

```
= axis
= array
= block
= chart
= collection
= command
= component
= component_folder
= component_reference
= component_relation
= connection
= domain
= edit_display
= file
= graph
= grid
= list
= image
= interface
= item_array
= menu
= method
= program
= record
= refresh_relation
= response_codes_definition
= source
= template
= unit_relation
= variable
= variable_list
= waveform
= wao_relation
```

A.6.3 AXIS

```
axis:
```

```
= 'AXIS' Identifier '{' axis_attribute_list '}'
= 'AXIS' Identifier '{' '}'
```

```
axis_attribute_list:
```

```
= axis_attribute
= axis_attribute_list axis_attribute
```

```
axis_attribute:
```

```
= constant_unit          /* O */
= help                    /* O */
= label                   /* O */
= axis_max_value          /* O */
= axis_min_value          /* O */
= axis_scaling            /* O */
```

```
axis_min_value:
```

```
= minimum_value
```

```
axis_max_value:
```

```

    = maximum_value

axis_scaling:
    = 'SCALING' axis_scaling_specifier

axis_scaling_specifier:
    = axis_scaling_keyword ';'
    = 'IF' '(' expr ')' '{' axis_scaling_specifier '}'
    = 'IF' '(' expr ')' '{' axis_scaling_specifier '}'
    'ELSE' '{' axis_scaling_specifier '}'
    = 'SELECT' '(' expr ')' '{' axis_scaling_selection_list '}'

axis_scaling_selection_list:
    = axis_scaling_selection
    = axis_scaling_selection_list axis_scaling_selection

axis_scaling_selection:
    = 'CASE' expr ':' axis_scaling_specifier
    = 'DEFAULT' ':' axis_scaling_specifier

axis_scaling_keyword:
    = 'LINEAR'
    = 'LOGARITHMIC'

```

A.6.4 BLOCK_A and BLOCK_B

```

block:
    = 'BLOCK' Identifier '{' block_attribute_list '}'

block_attribute_list:
    = block_attribute
    = block_attribute_list block_attribute

block_attribute:
    = block_a_characteristics /* M */
    = label /* M */
    = block_a_parameters /* M */
    = block_a_axis_items /* O */
    = block_a_chart_items /* O */
    = block_a_collection_items /* O */
    = block_a_edit_display_items /* O */
    = block_a_file_items /* O */
    = block_a_graph_items /* O */
    = block_a_grid_items /* O */
    = help /* O */
    = block_a_image_items /* O */
    = block_a_item_array_items /* O */
    = block_a_list_items /* O */
    = block_a_local_parameters /* O */
    = block_a_menu_items /* O */
    = block_a_method_items /* O */
    = block_a_parameter_lists /* O */
    = block_a_refresh_items /* O */
    = block_a_source_items /* O */
    = block_a_unit_items /* O */
    = block_a_wao_items /* O */
    = block_a_waveform_items /* O */
    = block_a_charts /* O */
    = block_a_lists /* O */
    = block_a_graphs /* O */
    = block_a_grids /* O */
    = block_a_menus /* O */
    = block_a_methods /* O */
    = block_b_number /* M */

```

```

    = block_b_type                                /* M */

block_a_characteristics:
    = 'CHARACTERISTICS' record_reference ';'

block_a_parameters:
    = 'PARAMETERS' '{' member_list '}'

member_list:
    = member
    = member_list member

block_a_axis_items:
    = 'AXIS_ITEMS' '{' axis_items_specifier_list '}'

axis_items_specifier_list:
    = axis_items_specifier
    = axis_items_specifier_list ',' axis_items_specifier

axis_items_specifier:
    = axis_reference

block_a_chart_items:
    = 'CHART_ITEMS' '{' chart_items_specifier_list '}'

chart_items_specifier_list:
    = chart_items_specifier
    = chart_items_specifier_list ',' chart_items_specifier

chart_items_specifier:
    = chart_reference

block_a_collection_items:
    = 'COLLECTION_ITEMS' '{' collection_items_specifier_list '}'

collection_items_specifier_list:
    = collection_items_specifier
    = collection_items_specifier_list ',' collection_items_specifier

collection_items_specifier:
    = collection_reference

block_a_edit_display_items:
    = 'EDIT_DISPLAY_ITEMS' '{' edit_display_items_specifier_list '}'

edit_display_items_specifier_list:
    = edit_display_items_specifier
    = edit_display_items_specifier_list ',' edit_display_items_specifier

edit_display_items_specifier:
    = edit_display_reference

block_a_file_items:
    = 'FILE_ITEMS' '{' file_items_specifier_list '}'

file_items_specifier_list:
    = file_items_specifier
    = file_items_specifier_list ',' file_items_specifier

file_items_specifier:
    = file_reference

block_a_graph_items:
    = 'GRAPH_ITEMS' '{' graph_items_specifier_list '}'

```



```
graph_items_specifier_list:
    = graph_items_specifier
    = graph_items_specifier_list ',' graph_items_specifier

graph_items_specifier:
    = graph_reference

block_a_grid_items:
    = 'GRID_ITEMS' '{' grid_items_specifier_list '}'

grid_items_specifier_list:
    = grid_items_specifier
    = grid_items_specifier_list ',' grid_items_specifier

grid_items_specifier:
    = grid_reference

block_a_image_items:
    = 'IMAGE_ITEMS' '{' image_items_specifier_list '}'

image_items_specifier_list:
    = image_items_specifier
    = image_items_specifier_list ',' image_items_specifier

image_items_specifier:
    = image_reference

block_a_item_array_items:
    = 'ITEM_ARRAY_ITEMS' '{' item_array_items_specifier_list '}'

item_array_items_specifier_list:
    = item_array_items_specifier
    = item_array_items_specifier_list ',' item_array_items_specifier

item_array_items_specifier:
    = item_array_reference

block_a_list_items:
    = 'LIST_ITEMS' '{' list_items_specifier_list '}'

list_items_specifier_list:
    = list_items_specifier
    = list_items_specifier_list ',' list_items_specifier

list_items_specifier:
    = list_reference

block_a_local_parameters:
    = 'LOCAL_PARAMETERS' '{' member_list '}'

block_a_menu_items:
    = 'MENU_ITEMS' '{' block_a_menu_items_specifier_list '}'

block_a_menu_items_specifier_list:
    = block_a_menu_items_specifier
    = block_a_menu_items_specifier_list ',' block_a_menu_items_specifier

block_a_menu_items_specifier:
    = menu_reference

block_a_method_items:
    = 'METHOD_ITEMS' '{' method_items_specifier_list '}'
```

```

method_items_specifier_list:
    = method_items_specifier
    = method_items_specifier_list ',' method_items_specifier

method_items_specifier:
    = method_reference

block_a_parameter_lists:
    = 'PARAMETER_LISTS' '{' member_list '}'

block_a_refresh_items:
    = 'REFRESH_ITEMS' '{' refresh_items_specifier_list '}'

refresh_items_specifier_list:
    = refresh_items_specifier
    = refresh_items_specifier_list ',' refresh_items_specifier

refresh_items_specifier:
    = refresh_reference

block_a_source_items:
    = 'SOURCE_ITEMS' '{' source_items_specifier_list '}'

source_items_specifier_list:
    = source_items_specifier
    = source_items_specifier_list ',' source_items_specifier

source_items_specifier:
    = source_reference

block_a_unit_items:
    = 'UNIT_ITEMS' '{' unit_items_specifier_list '}'

unit_items_specifier_list:
    = unit_items_specifier
    = unit_items_specifier_list ',' unit_items_specifier

unit_items_specifier:
    = unit_reference

block_a_wao_items:
    = 'WRITE_AS_ONE_ITEMS' '{' wao_items_specifier_list '}'

wao_items_specifier_list:
    = wao_items_specifier
    = wao_items_specifier_list ',' wao_items_specifier

wao_items_specifier:
    = wao_reference

block_a_waveform_items:
    = 'WAVEFORM_ITEMS' '{' waveform_items_specifier_list '}'

waveform_items_specifier_list:
    = waveform_items_specifier
    = waveform_items_specifier_list ',' waveform_items_specifier

waveform_items_specifier:
    = waveform_reference

block_a_charts:
    = 'CHARTS' '{' member_list '}'

```

```

block_a_lists:
    = 'LISTS' '{' member_list '}'

block_a_graphs:
    = 'GRAPHS' '{' member_list '}'

block_a_grids:
    = 'GRIDS' '{' member_list '}'

block_a_menus:
    = 'MENUS' '{' member_list '}'

block_a_methods:
    = 'METHODS' '{' member_list '}'

block_b_number:
    = 'NUMBER' expr ';'

block_b_type:
    = 'TYPE' 'PHYSICAL' ';'
    = 'TYPE' 'TRANSDUCER' ';'
    = 'TYPE' 'FUNCTION' ';'

```

A.6.5 CHART

```

chart:
    = 'CHART' Identifier '{' chart_attribute_list '}'

chart_attribute_list:
    = chart_attribute
    = chart_attribute_list chart_attribute

chart_attribute:
    = members /* M */
    = chart_cycle_time /* O */
    = help /* O */
    = height /* O */
    = label /* O */
    = chart_length /* O */
    = chart_type /* O */
    = validity /* O */
    = width /* O */

chart_cycle_time:
    = 'CYCLE_TIME' expr_specifier

chart_length:
    = 'LENGTH' expr_specifier

chart_type:
    = 'TYPE' chart_type_specifier

chart_type_specifier:
    = chart_type_keyword ';'
    = 'IF' '(' expr ')' '{' chart_type_specifier '}'
    = 'IF' '(' expr ')' '{' chart_type_specifier '}'
    = 'ELSE' '{' chart_type_specifier '}'
    = 'SELECT' '(' expr ')' '{' chart_type_selection_list '}'

chart_type_selection_list:
    = chart_type_selection
    = chart_type_selection_list chart_type_selection

```

```
chart_type_selection:
    = 'CASE' expr ':' chart_type_specifier
    = 'DEFAULT' ':' chart_type_specifier
```

```
chart_type_keyword:
    = 'GAUGE'
    = 'HORIZONTAL_BAR'
    = 'SCOPE'
    = 'STRIP'
    = 'SWEEP'
    = 'VERTICAL_BAR'
```

A.6.6 COLLECTION

```
collection:
    = 'COLLECTION' Identifier '{' collection_attribute_list '}'
    = 'COLLECTION' 'OF' item_type Identifier '{'
        collection_attribute_list '}'
```

```
item_type:
    = 'ARRAY'
    = 'AXIS'
    = 'BLOCK'
    = 'CHART'
    = 'COLLECTION' 'OF' item_type
    = 'COMMAND'
    = 'COMPONENT'
    = 'COMPONENT_FOLDER'
    = 'COMPONENT_REFERENCE'
    = 'COMPONENT_RELATION'
    = 'CONNECTION'
    = 'DOMAIN'
    = 'EDIT_DISPLAY'
    = 'FILE'
    = 'GRAPH'
    = 'GRID'
    = 'IMAGE'
    = 'INTERFACE'
    = 'ITEM_ARRAY' 'OF' item_type
    = 'LIST'
    = 'MENU'
    = 'METHOD'
    = 'PROGRAM'
    = 'RECORD'
    = 'REFRESH'
    = 'RESPONSE_CODES'
    = 'SOURCE'
    = 'TEMPLATE'
    = 'UNIT'
    = 'VARIABLE'
    = 'VARIABLE_LIST'
    = 'WAVEFORM'
    = 'WRITE_AS_ONE'
```

```
collection_attribute_list:
    = collection_attribute
    = collection_attribute_list collection_attribute
```

```
collection_attribute:
    = members                /* M */
    = help                   /* O */
    = label                  /* O */
    = validity               /* O */
```

A.6.7 COMMAND

```

command:
    = 'COMMAND' Identifier '{' command_attribute_list '}'
    = 'COMMAND' Identifier '{' '}'

command_attribute_list:
    = command_attribute
    = command_attribute_list command_attribute

command_attribute:
    = command_operation          /* M */
    = command_transaction        /* M */
    = command_index              /* O */
    = command_block              /* O */
    = command_number             /* O */
    = command_slot               /* O */
    = command_sub_slot           /* O */
    = command_connection         /* O */
    = command_header             /* O */
    = command_module             /* O */
    = response_codes             /* O */

command_operation:
    = 'OPERATION' operation ';'

operation:
    = 'READ'
    = 'WRITE'
    = 'COMMAND'
    = 'DATA_EXCHANGE'
    = string

command_transaction:
    = 'TRANSACTION' '{' transaction_attribute_list '}'
    = 'TRANSACTION' Integer '{' transaction_attribute_list '}'

transaction_attribute_list:
    = transaction_attribute
    = transaction_attribute_list transaction_attribute

transaction_attribute:
    = request                    /* M */
    = reply                      /* M */
    = 'RESPONSE_CODES' '(' response_code_reference ')' /* O */

request:
    = 'REQUEST' '{' data_items_specifier_list '}'
    = 'REQUEST' '{' '}'

reply:
    = 'REPLY' '{' data_items_specifier_list '}'
    = 'REPLY' '{' '}'

data_items_specifier_list:
    = data_items_specifier
    = data_items_specifier_list ',' data_items_specifier

data_items_specifier:
    = Integer
    = variable_reference
    = variable_reference '<' Integer '>'
    = variable_reference '(' data_items_qualifier_list ')'

```

```

= variable_reference '<' Integer '>' '(' data_items_qualifier_list ')'
= array_reference
= array_reference '(' data_items_qualifier_list ')'
= collection_reference
= collection_reference '(' data_items_qualifier_list ')'
= item_array_reference
= item_array_reference '(' data_items_qualifier_list ')'
= list_reference
= list_reference '(' data_items_qualifier_list ')'

data_items_qualifier_list:
= data_items_qualifier
= data_items_qualifier_list ',' data_items_qualifier

data_items_qualifier:
= 'INDEX'
= 'INFO'

command_index:
= 'INDEX' index_specifier

index_specifier:
= index_items ';'
= 'IF' '(' expr ')' '{' index_specifier '}'
= 'IF' '(' expr ')' '{' index_specifier '}'
'ELSE' '{' index_specifier '}'
= 'SELECT' '(' expr ')' '{' index_selection_list '}'

index_items:
= expr
= variable_reference

index_selection_list:
= index_selection
= index_selection_list index_selection

index_selection:
= 'CASE' expr ':' index_specifier
= 'DEFAULT' ':' index_specifier

command_block:
= 'BLOCK' block_specifier

block_specifier:
= block_b_reference ';'
= 'IF' '(' expr ')' '{' block_specifier '}'
= 'IF' '(' expr ')' '{' block_specifier '}'
'ELSE' '{' block_specifier '}'
= 'SELECT' '(' expr ')' '{' block_selection_list '}'

block_selection_list:
= block_selection
= block_selection_list block_selection

block_selection:
= 'CASE' expr ':' block_specifier
= 'DEFAULT' ':' block_specifier

command_number:
= 'NUMBER' number_specifier

number_specifier:
= number_items ';'

```

```

    = 'IF' '(' expr ')' '{' number_specifier '}'
    = 'IF' '(' expr ')' '{' number_specifier '}'
    'ELSE' '{' number_specifier '}'
    = 'SELECT' '(' expr ')' '{' number_selection_list '}'

number_items:
    = Integer
    = variable_reference

number_selection_list:
    = number_selection
    = number_selection_list number_selection

number_selection:
    = 'CASE' expr ':' number_specifier
    = 'DEFAULT' ':' number_specifier

command_slot:
    = 'SLOT' slot_specifier

slot_specifier:
    = slot_items ';'
    = 'IF' '(' expr ')' '{' slot_specifier '}'
    = 'IF' '(' expr ')' '{' slot_specifier '}'
    'ELSE' '{' slot_specifier '}'
    = 'SELECT' '(' expr ')' '{' slot_selection_list '}'

slot_items:
    = expr
    = variable_reference

slot_selection_list:
    = slot_selection
    = slot_selection_list slot_selection

slot_selection:
    = 'CASE' expr ':' slot_specifier
    = 'DEFAULT' ':' slot_specifier

command_sub_slot:
    = 'SUB_SLOT' slot_specifier

command_connection:
    = 'CONNECTION' connection_specifier

connection_specifier:
    = connection_reference ';'
    = 'IF' '(' expr ')' '{' connection_specifier '}'
    = 'IF' '(' expr ')' '{' connection_specifier '}'
    'ELSE' '{' connection_specifier '}'
    = 'SELECT' '(' expr ')' '{' connection_selection_list '}'

connection_selection_list:
    = connection_selection
    = connection_selection_list connection_selection

connection_selection:
    = 'CASE' expr ':' connection_specifier
    = 'DEFAULT' ':' connection_specifier

command_header:
    = 'HEADER' header_specifier

```

```
header_specifier:
    = string ';'
    = 'IF' '(' expr ')' '{' header_specifier '}'
    = 'IF' '(' expr ')' '{' header_specifier '}'
      'ELSE' '{' header_specifier '}'
    = 'SELECT' '(' expr ')' '{' header_selection_list '}'
```

```
header_selection_list:
    = header_selection
    = header_selection_list header_selection
```

```
header_selection:
    = 'CASE' expr ':' header_specifier
    = 'DEFAULT' ':' header_specifier
```

```
command_module:
    = 'MODULE' module_specifier
```

```
module_specifier:
    = Identifier ';'
    = 'IF' '(' expr ')' '{' module_specifier '}'
    = 'IF' '(' expr ')' '{' module_specifier '}'
      'ELSE' '{' module_specifier '}'
    = 'SELECT' '(' expr ')' '{' module_selection_list '}'
```

```
module_selection_list:
    = module_selection
    = module_selection_list module_selection
```

```
module_selection:
    = 'CASE' expr ':' module_specifier
    = 'DEFAULT' ':' module_specifier
```

A.6.8 COMPONENT

```
component:
    = 'COMPONENT' Identifier '{' component_attribute_list '}'
```

```
component_attribute_list:
    = component_attribute
    = component_attribute_list component_attribute
```

```
component_attribute:
    = label /* M */
    = can_delete /* O */
    = check_configuration /* O */
    = classification /* O */
    = component_parent /* O */
    = component_path /* O */
    = component_relations /* O */
    = declaration /* O */
    = detect_method /* O */
    = edd /* O */
    = help /* O */
    = initial_values /* O */
    = redundancy /* O */
    = protocol /* O */
    = scan_method /* O */
    = scan_list /* O */
    = supplied_interface /* O */
```

```
can_delete:
    = 'CAN_DELETE' boolean_specifier ';'
```



```

check_configuration:
    = 'CHECK_CONFIGURATION' expr_specifier

classification:
    = 'CLASSIFICATION' classification_specifier ';'

classification_specifier:
    = 'ACTUATOR'
    = 'ACTUATOR_ELECTRO_PNEUMATIC'
    = 'ACTUATOR_ELECTRIC'
    = 'ACTUATOR_HYDRAULIC'
    = 'CONVERTER'
    = 'CONTROLLER'
    = 'DISCRETE_IN'
    = 'DISCRETE_OUT'
    = 'ELECTRICAL_DISTRIBUTION'
    = 'ELECTRICAL_DISTRIBUTION_POWER_MONITORING'
    = 'FREQUENCY_CONVERTER'
    = 'INDICATOR'
    = 'NETWORK_COMPONENT'
    = 'NETWORK_COMPONENT_WIRELESSADAPTER'
    = 'REMOTEIO'
    = 'RECORDER'
    = 'SENSOR'
    = 'SENSOR_ACOUSTIC'
    = 'SENSOR_ANALYTIC'
    = 'SENSOR_ANALYTIC_GAS'
    = 'SENSOR_ANALYTIC_LIQUID'
    = 'SENSOR_CONCENTRATION'
    = 'SENSOR_DENSITY'
    = 'SENSOR_DENSITY_RADIOMETRIC'
    = 'SENSOR_FLOW'
    = 'SENSOR_FLOW_CORIOLIS'
    = 'SENSOR_FLOW_ELECTRO_MAGNETIC'
    = 'SENSOR_FLOW_MECHANICAL'
    = 'SENSOR_FLOW_RADIOMETRIC'
    = 'SENSOR_FLOW_THERMAL'
    = 'SENSOR_FLOW_ULTRASONIC'
    = 'SENSOR_FLOW_VORTEX_COUNTER'
    = 'SENSOR_LEVEL'
    = 'SENSOR_LEVEL_BUOYANCY'
    = 'SENSOR_LEVEL_CAPACITIVE'
    = 'SENSOR_LEVEL_ECHO'
    = 'SENSOR_LEVEL_HYDROSTATIC'
    = 'SENSOR_LEVEL_RADIOMETRIC'
    = 'SENSOR_MULTIVARIABLE'
    = 'SENSOR_POSITION'
    = 'SENSOR_PRESSURE'
    = 'SENSOR_TEMPERATURE'
    = 'SEMICONDUCTOR'
    = 'SWITCHGEAR'
    = 'UNIVERSAL'
    = string

component_path:
    = 'COMPONENT_PATH' string ';'

component_parent:
    = 'COMPONENT_PARENT' reference ';'

component_relations:
    = 'COMPONENT_RELATIONS' '{' component_specifier_list '}'

```

```

component_specifier_list:
    = component_specifier
    = component_specifier_list ',' component_specifier

component_specifier:
    = Identifier
    = 'IF' '(' expr ')' '{' component_specifier_list '}'
    = 'IF' '(' expr ')' '{' component_specifier_list '}'
    = 'ELSE' '{' component_specifier_list '}'
    = 'SELECT' '(' expr ')' '{' component_selection_list '}'

component_selection_list:
    = component_selection
    = component_selection_list component_selection

component_selection:
    = 'CASE' expr ':' component_specifier_list
    = 'DEFAULT' ':' component_specifier_list

declaration:
    = 'DECLARATION' '{' item_list '}'

item_list:
    = item
    = item_list item

detect_method:
    = 'DETECT' method_reference ';'

edd:
    = 'EDD' string ';'

protocol:
    = 'PROTOCOL' protocol_specifier ';'

protocol_specifier:
    = 'PROFIBUS_DP'
    = 'PROFIBUS_PA'
    = 'PROFINET'
    = 'HART'
    = 'FF'
    = string

redundancy:
    = 'REDUNDANCY' expr_specifier

initial_values:
    = 'INITIAL_VALUES' '{' initial_value_list '}'

initial_value_list:
    = initial_value_specifier
    = initial_value_list initial_value_specifier

initial_value_specifier:
    = reference_without_call '=' expr_specifier

scan_method:
    = 'SCAN' method_reference ';'

scan_list:
    = 'SCAN_LIST' reference_without_call ';'

```

```
supplied_interface:
    = 'SUPPLIED_INTERFACE' '{' interface_list '}'
```

A.6.9 COMPONENT_FOLDER

```
component_folder:
    = 'COMPONENT_FOLDER' Identifier '{' component_folder_attribute_list '}'
```

```
component_folder_attribute_list:
    = component_folder_attribute
    = component_folder_attribute_list component_folder_attribute
```

```
component_folder_attribute:
    = label                      /* M */
    = classification             /* O */
    = component_parent           /* O */
    = component_path             /* O */
    = help                       /* O */
    = protocol                   /* O */
```

A.6.10 COMPONENT_REFERENCE

```
component_reference:
    = 'COMPONENT_REFERENCE' Identifier '{'
        component_reference_attribute_list '}'
```

```
component_reference_attribute_list:
    = component_reference_attribute
    = component_reference_attribute_list component_reference_attribute
```

```
component_reference_attribute:
    = classification             /* O */
    = component_parent           /* O */
    = component_path             /* O */
    = device_revision_attribute  /* O */
    = device_type_attribute      /* O */
    = manufacturer_attribute     /* O */
    = protocol                   /* O */
```

```
device_revision_attribute:
    = device_revision ';' '
```

```
device_type_attribute:
    = device_type ';' '
```

```
manufacturer_attribute:
    = manufacturer ';' '
```

A.6.11 COMPONENT_RELATION

```
component_relation:
    = 'COMPONENT_RELATION' Identifier '{'
        component_relation_attribute_list '}'
```

```
component_relation_attribute_list:
    = component_relation_attribute
    = component_relation_attribute_list component_relation_attribute
```

```
component_relation_attribute:
    = components                 /* M */
    = label                      /* M */
    = relation_type              /* M */
    = addressing                 /* O */
    = help                       /* O */
    = maximum_number             /* O */
    = minimum_number             /* O */
```

```

= required_interface          /* 0 */
= supplied_interface          /* 0 */

components:
= 'COMPONENTS' '{' component_relation_reference_list '}'

component_relation_reference_list:
= component_relation_reference
= component_relation_reference_list ','
  component_relation_reference

component_relation_reference:
= component_relation_reference_specifier
= 'IF' '(' expr ')' '{' component_relation_reference_list '}'
= 'IF' '(' expr ')' '{' component_relation_reference_list '}'
  'ELSE' '{' component_relation_reference_list '}'
= 'SELECT' '(' expr ')' '{' component_relation_selection_list '}'

component_relation_selection_list:
= component_relation_selection
= component_relation_selection_list component_relation_selection

component_relation_selection:
= 'CASE' expr ':' component_relation_reference_list
= 'DEFAULT' ':' component_relation_reference_list

component_relation_reference_specifier:
= Identifier
= Identifier '{' component_relation_arg_list '}'

component_relation_arg_list:
= component_relation_arg
= component_relation_arg_list component_relation_arg

component_relation_arg:
= minimum_number
= maximum_number
= auto_create
= filter
= required_ranges

auto_create:
= 'AUTO_CREATE' expr_specifier

filter:
= 'FILTER' '(' expr ')'

required_ranges:
= 'REQUIRED_RANGES' '{' required_ranges_list '}'

required_ranges_list:
= required_range_specifier
= required_ranges_list ';' required_range_specifier

required_range_specifier:
= variable_reference '{' variable_range_arg_list '}'

variable_range_arg_list:
= variable_range_arg
= variable_range_arg_list variable_range_arg

variable_range_arg:

```

```

    = maximum_value
    = maximum_value_n
    = minimum_value
    = minimum_value_n

relation_type:
    = 'RELATION_TYPE' relation_type_specifier ';'

relation_type_specifier:
    = 'PARENT_COMPONENT'
    = 'CHILD_COMPONENT'
    = 'SIBLING_COMPONENT'
    = 'NEXT_COMPONENT'
    = 'PREV_COMPONENT'

addressing:
    = 'ADDRESSING' '{' addressing_list '}'

addressing_list:
    = variable_reference
    = addressing_list ',' variable_reference

maximum_number:
    = 'MAXIMUM_NUMBER' expr_specifier

minimum_number:
    = 'MINIMUM_NUMBER' expr_specifier

required_interface:
    = 'REQUIRED_INTERFACE' '{' interface_list '}'

interface_list:
    = interface_reference
    = interface_list ',' interface_reference

interface_reference:
    = Identifier

A.6.12 CONNECTION

connection:
    = 'CONNECTION' Identifier '{' connection_attribute_list '}'

connection_attribute_list:
    = connection_attribute
    = connection_attribute_list connection_attribute

connection_attribute:
    = appinstance                                     /* M */

appinstance:
    = 'APPINSTANCE' Integer ';'

A.6.13 DOMAIN

domain:
    = 'DOMAIN' Identifier '{' domain_attribute_list '}'
    = 'DOMAIN' Identifier '{' '}'

domain_attribute_list:
    = domain_attribute
    = domain_attribute_list domain_attribute

domain_attribute:
    = handling                                     /* O */

```

```
= response_codes                                /* 0 */
```

A.6.14 EDIT_DISPLAY

```
edit_display:
    = 'EDIT_DISPLAY' Identifier '{' edit_display_attribute_list '}'
```

```
edit_display_attribute_list:
    = edit_display_attribute
    = edit_display_attribute_list edit_display_attribute
```

```
edit_display_attribute:
    = edit_items                                /* M */
    = label                                    /* M */
    = display_items                            /* O */
    = post_edit_actions                        /* O */
    = pre_edit_actions                         /* O */
```

```
edit_items:
    = 'EDIT_ITEMS' '{' edit_items_specifier_list '}'
```

```
edit_items_specifier_list:
    = edit_items_specifier
    = edit_items_specifier_list edit_items_specifier
```

```
edit_items_specifier:
    = edit_item_list
    = 'IF' '(' expr ')' '{' edit_items_specifier_list '}'
    = 'IF' '(' expr ')' '{' edit_items_specifier_list '}'
    = 'ELSE' '{' edit_items_specifier_list '}'
    = 'SELECT' '(' expr ')' '{' edit_items_selection_list '}'
```

```
edit_items_selection_list:
    = edit_items_selection
    = edit_items_selection_list edit_items_selection
```

```
edit_items_selection:
    = 'CASE' expr ':' edit_items_specifier_list
    = 'DEFAULT' ':' edit_items_specifier_list
```

```
edit_item_list:
    = edit_item
    = edit_item_list ',' edit_item
```

```
edit_item:
    = reference_without_call
```

```
display_items:
    = 'DISPLAY_ITEMS' '{' display_items_specifier_list '}'
```

```
display_items_specifier_list:
    = display_items_specifier
    = display_items_specifier_list display_items_specifier
```

```
display_items_specifier:
    = display_item_list
    = 'IF' '(' expr ')' '{' display_items_specifier_list '}'
    = 'IF' '(' expr ')' '{' display_items_specifier_list '}'
    = 'ELSE' '{' display_items_specifier_list '}'
    = 'SELECT' '(' expr ')' '{' display_items_selection_list '}'
```

```
display_items_selection_list:
    = display_items_selection
    = display_items_selection_list display_items_selection
```

```

display_items_selection:
    = 'CASE' expr ':' display_items_specifier_list
    = 'DEFAULT' ':' display_items_specifier_list

display_item_list:
    = display_item
    = display_item_list ',' display_item

display_item:
    = variable_reference

```

A.6.15 FILE

```

file:
    = 'FILE' Identifier '{' file_attribute_list '}'

file_attribute_list:
    = file_attribute
    = file_attribute_list file_attribute

file_attribute:
    = members /* M */
    = label /* O */
    = help /* O */

```

A.6.16 GRAPH

```

graph:
    = 'GRAPH' Identifier '{' graph_attribute_list '}'

graph_attribute_list:
    = graph_attribute
    = graph_attribute_list graph_attribute

graph_attribute:
    = members /* M */
    = label /* O */
    = help /* O */
    = height /* O */
    = graph_cycle_time /* O */
    = validity /* O */
    = width /* O */
    = graph_x_axis /* O */

graph_cycle_time:
    = 'CYCLE_TIME' expr_specifier

graph_x_axis:
    = 'X_AXIS' axis_specifier

```

A.6.17 GRID

```

grid:
    = 'GRID' Identifier '{' grid_attribute_list '}'

grid_attribute_list:
    = grid_attribute
    = grid_attribute_list grid_attribute

grid_attribute:
    = grid_vectors /* M */
    = handling /* O */
    = height /* O */
    = help /* O */
    = label /* O */

```

```

= grid_orientation          /* O */
= validity                  /* O */
= width                     /* O */

grid_vectors:
= 'VECTORS' '{' vector_specifier_list '}'

vector_specifier_list:
= vector_specifier
= vector_specifier_list ',' vector_specifier

vector_specifier:
= vector
= 'IF' '(' expr ')' '{' vector_specifier_list '}'
= 'IF' '(' expr ')' '{' vector_specifier_list '}'
  'ELSE' '{' vector_specifier_list '}'
= 'SELECT' '(' expr ')' '{' vector_selection_list '}'

vector_selection_list:
= vector_selection
= vector_selection_list vector_selection

vector_selection:
= 'CASE' expr ':' vector_specifier_list
= 'DEFAULT' ':' vector_specifier_list

vector:
= '{' string ',' vector_values_list '}'

vector_values_list:
= vector_values
= vector_values_list ',' vector_values

vector_values:
= reference_without_call
= Integer
= Real
= string

grid_orientation:
= 'ORIENTATION' orientation_specifier ';'

orientation_specifier:
= 'HORIZONTAL'
= 'VERTICAL'

```

A.6.18 IMAGE

```

image:
= 'IMAGE' Identifier '{' image_attribute_list '}'

image_attribute_list:
= image_attribute
= image_attribute_list image_attribute

image_attribute:
= image_path          /* M */
= label               /* O */
= help                /* O */
= image_link          /* O */
= validity            /* O */

image_path:
= 'PATH' file_name_specifier

```



```

file_name_specifier:
    = string_literal ';'
    = 'IF' '(' expr ')' '{' file_name_specifier '}'
    = 'IF' '(' expr ')' '{' file_name_specifier '}'
    = 'ELSE' '{' file_name_specifier '}'
    = 'SELECT' '(' expr ')' '{' file_name_selection_list '}'

file_name_selection_list:
    = file_name_selection
    = file_name_selection_list file_name_selection

file_name_selection:
    = 'CASE' expr ':' file_name_specifier
    = 'DEFAULT' ':' file_name_specifier

image_link:
    = 'LINK' link_specifier

link_specifier:
    = reference_without_call ';'
    = 'IF' '(' expr ')' '{' link_specifier '}'
    = 'IF' '(' expr ')' '{' link_specifier '}'
    = 'ELSE' '{' link_specifier '}'
    = 'SELECT' '(' expr ')' '{' link_selection_list '}'

link_selection_list:
    = link_selection
    = link_selection_list link_selection

link_selection:
    = 'CASE' expr ':' link_specifier
    = 'DEFAULT' ':' link_specifier

```

A.6.19 INTERFACE

```

interface:
    = 'INTERFACE' Identifier '{' interface_attribute_list '}'

interface_attribute_list:
    = interface_attribute
    = interface_attribute_list interface_attribute

interface_attribute:
    = declaration /* M */
    = label /* M */
    = help /* O */

```

A.6.20 LIST

```

list:
    = 'LIST' Identifier '{' list_attribute_list '}'

list_attribute_list:
    = list_attribute
    = list_attribute_list list_attribute

list_attribute:
    = list_type /* M */
    = list_capacity /* O */
    = list_count /* O */
    = help /* O */
    = label /* O */
    = validity /* O */

```

```
list_type:
    = 'TYPE' list_type_specifier ';'

list_type_specifier:
    = reference_without_call
```

```
list_capacity:
    = 'CAPACITY' Integer ';'

list_count:
    = 'COUNT' expr_specifier
```

A.6.21 IMPORT

```
imported_description:
    = 'IMPORT' identification '{' imports '}'
    = 'IMPORT' identification '{' imports redefinitions '}'
    = 'IMPORT' '""' filename '""'
    = 'IMPORT' '<' filename '>'
```

```
imports:
    = 'EVERYTHING' ';'
    = item_import_list
```

```
item_import_list:
    = item_import
    = item_import_list item_import
```

```
item_import:
    = item_import_by_name ';'
    = item_import_by_type ';'

item_import_by_name:
    = item_type Identifier
    = Identifier
```

```
item_import_by_type:
    = import_item_type
    = item_import_by_type '&' import_item_type
```

```
import_item_type:
    = 'ARRAYS'
    = 'AXES'
    = 'BLOCKS'
    = 'CHARTS'
    = 'COLLECTIONS'
    = 'COMMANDS'
    = 'COMPONENTS'
    = 'COMPONENT_FOLDERS'
    = 'COMPONENT_REFERENCES'
    = 'COMPONENT_RELATIONS'
    = 'CONNECTIONS'
    = 'DOMAINS'
    = 'EDIT_DISPLAYS'
    = 'FILES'
    = 'GRAPHS'
    = 'GRIDS'
    = 'IMAGES'
    = 'INTERFACES'
    = 'ITEM_ARRAYS'
    = 'LISTS'
    = 'MENUS'
    = 'METHODS'
    = 'PROGRAMS'
```

```

= 'RECORDS'
= 'REFRESHES'
= 'RELATIONS'
= 'RESPONSE_CODES'
= 'SOURCES'
= 'TEMPLATES'
= 'UNITS'
= 'VARIABLES'
= 'VARIABLE_LISTS'
= 'WRITE_AS_ONES'
= 'WAVEFORMS'

```

redefinitions:

```

= 'REDEFINITIONS' '{' redefinition_list '}'

```

redefinition_list:

```

= redefinition
= redefinition_list redefinition

```

redefinition:

```

= array_redefinition
= axis_redefinition
= block_redefinition
= chart_redefinition
= collection_redefinition
= command_redefinition
= component_redefinition
= component_folder_redefinition
= component_reference_redefinition
= component_relation_redefinition
= connection_redefinition
= domain_redefinition
= edit_display_redefinition
= file_redefinition
= graph_redefinition
= grid_redefinition
= image_redefinition
= interface_redefinition
= item_array_redefinition
= list_redefinition
= menu_redefinition
= method_redefinition
= program_redefinition
= record_redefinition
= refresh_relation_redefinition
= response_codes_definition_redefinition
= source_redefinition
= template_redefinition
= unit_relation_redefinition
= variable_redefinition
= variable_list_redefinition
= wao_relation_redefinition
= waveform_redefinition

```

filename:

```

= string_literal

```

A.6.22 LIKE

like:

```

= Identifier 'LIKE' Identifier
  '{' attribute_redefinition_list '}'
= Identifier 'LIKE' 'ARRAY' Identifier

```

```
'{' array_attribute_redefinition_list '}'
= Identifier 'LIKE' 'AXIS' Identifier
'{' axis_attribute_redefinition_list '}'
= Identifier 'LIKE' 'BLOCK' Identifier
'{' block_attribute_redefinition_list '}'
= Identifier 'LIKE' 'CHART' Identifier
'{' chart_attribute_redefinition_list '}'
= Identifier 'LIKE' 'COLLECTION' 'OF' item_type Identifier
'{' collection_attribute_redefinition_list '}'
= Identifier 'LIKE' 'COMMAND' Identifier
'{' command_attribute_redefinition_list '}'
= Identifier 'LIKE' 'COMPONENT' Identifier
'{' component_attribute_redefinition_list '}'
= Identifier 'LIKE' 'COMPONENT_FOLDER' Identifier
'{' component_folder_attribute_redefinition_list '}'
= Identifier 'LIKE' 'COMPONENT_REFERENCE' Identifier
'{' component_reference_attribute_redefinition_list '}'
= Identifier 'LIKE' 'COMPONENT_RELATION' Identifier
'{' component_relation_attribute_redefinition_list '}'
= Identifier 'LIKE' 'CONNECTION' Identifier
'{' connection_attribute_redefinition_list '}'
= Identifier 'LIKE' 'DOMAIN' Identifier
'{' domain_attribute_redefinition_list '}'
= Identifier 'LIKE' 'EDIT_DISPLAY' Identifier
'{' edit_display_attribute_redefinition_list '}'
= Identifier 'LIKE' 'FILE' Identifier
'{' file_attribute_redefinition_list '}'
= Identifier 'LIKE' 'GRAPH' Identifier
'{' graph_attribute_redefinition_list '}'
= Identifier 'LIKE' 'GRID' Identifier
'{' grid_attribute_redefinition_list '}'
= Identifier 'LIKE' 'IMAGE' Identifier
'{' image_attribute_redefinition_list '}'
= Identifier 'LIKE' 'INTERFACE' Identifier
'{' interface_attribute_redefinition_list '}'
= Identifier 'LIKE' 'ITEM_ARRAY' 'OF' item_type
Identifier
'{' item_array_attribute_redefinition_list '}'
= Identifier 'LIKE' 'LIST' Identifier
'{' list_attribute_redefinition_list '}'
= Identifier 'LIKE' 'MENU' Identifier
'{' menu_attribute_redefinition_list '}'
= Identifier 'LIKE' 'METHOD' Identifier
'{' method_attribute_redefinition_list '}'
= Identifier 'LIKE' 'PROGRAM' Identifier
'{' program_attribute_redefinition_list '}'
= Identifier 'LIKE' 'RECORD' Identifier
'{' record_attribute_redefinition_list '}'
= Identifier 'LIKE' 'REFRESH' Identifier
'{' refresh_relation_left ':' refresh_relation_right '}'
= Identifier 'LIKE' 'RESPONSE_CODES' Identifier
'{' response_code_redefinition_list '}'
= Identifier 'LIKE' 'SOURCE' Identifier
'{' source_attribute_redefinition_list '}'
= Identifier 'LIKE' 'TEMPLATE' Identifier
'{' template_attribute_redefinition_list '}'
= Identifier 'LIKE' 'UNIT' Identifier
'{' unit_relation_left ':' unit_relation_right '}'
= Identifier 'LIKE' 'VARIABLE' Identifier
'{' variable_attribute_redefinition_list '}'
= Identifier 'LIKE' 'VARIABLE_LIST' Identifier
'{' variable_list_attribute_redefinition_list '}'
= Identifier 'LIKE' 'WAVEFORM' Identifier
```

```
'{' waveform_attribute_redefinition_list '}'
= Identifier 'LIKE' 'WRITE_AS_ONE' Identifier
'{' wao_specifier '}'
```

```
attribute_redefinition_list:
= attribute_redefinition
= attribute_redefinition_list attribute_redefinition
```

```
attribute_redefinition:
= array_attribute_redefinition
= axis_attribute_redefinition
= block_attribute_redefinition
= chart_attribute_redefinition
= collection_attribute_redefinition
= command_attribute_redefinition
= component_attribute_redefinition
= component_folder_attribute_redefinition
= component_reference_attribute_redefinition
= component_relation_attribute_redefinition
= connection_attribute_redefinition
= domain_attribute_redefinition
= edit_display_attribute_redefinition
= file_attribute_redefinition
= graph_attribute_redefinition
= grid_attribute_redefinition
= image_attribute_redefinition
= interface_attribute_redefinition
= item_array_attribute_redefinition
= list_attribute_redefinition
= menu_attribute_redefinition
= method_attribute_redefinition
= program_attribute_redefinition
= record_attribute_redefinition
= source_attribute_redefinition
= template_attribute_redefinition
= variable_attribute_redefinition
= variable_list_attribute_redefinition
= waveform_attribute_redefinition
```

A.6.23 MENU

```
menu:
= 'MENU' Identifier '{' menu_attribute_list '}'
```

```
menu_attribute_list:
= menu_attribute
= menu_attribute_list menu_attribute
```

```
menu_attribute:
= menu_items /* M */
= label /* M */
= menu_access /* O */
= help /* O */
= post_edit_actions /* O */
= post_read_actions /* O */
= post_write_actions /* O */
= pre_edit_actions /* O */
= pre_read_actions /* O */
= pre_write_actions /* O */
= menu_purpose /* O */
= menu_role /* O */
= menu_style /* O */
= validity /* O */
```

```

menu_items:
    = 'ITEMS' '{' '}'
    = 'ITEMS' '{' menu_item_specifier_list '}'

menu_item_specifier_list:
    = menu_item_specifier
    = menu_item_specifier_list menu_item_specifier

menu_item_specifier:
    = menu_item_list
    = 'IF' '(' expr ')' '{' menu_item_specifier_list '}'
    = 'IF' '(' expr ')' '{' menu_item_specifier_list '}'
    = 'ELSE' '{' menu_item_specifier_list '}'
    = 'SELECT' '(' expr ')' '{' menu_item_selection_list '}'

menu_item_selection_list:
    = menu_item_selection
    = menu_item_selection_list menu_item_selection

menu_item_selection:
    = 'CASE' expr ':' menu_item_specifier_list
    = 'DEFAULT' ':' menu_item_specifier_list

menu_item_list:
    = menu_item
    = menu_item_list ',' menu_item

menu_item:
    = reference_without_call
    = reference_without_call menu_item_args
    = 'SEPARATOR'
    = 'COLUMNBREAK'
    = 'ROWBREAK'

menu_item_args:
    = '(' argument_list ')'
    = '(' 'REVIEW' ')'
    = '(' variable_qualifier_list ')'
    = '(' image_qualifier_list ')'

variable_qualifier_list:
    = variable_qualifier
    = variable_qualifier_list ',' variable_qualifier

variable_qualifier:
    = 'DISPLAY_VALUE'
    = 'READ_ONLY'
    = 'HIDDEN'
    = 'NO_LABEL'
    = 'NO_UNIT'

image_qualifier_list:
    = image_qualifier
    = image_qualifier_list ',' image_qualifier

image_qualifier:
    = 'INLINE'

menu_access:
    = 'ACCESS' 'ONLINE' ';'
    = 'ACCESS' 'OFFLINE' ';'

menu_purpose:

```

```

    = 'PURPOSE' menu_purpose_specifier

menu_purpose_specifier:
    = menu_purpose_list ';'
    = 'IF' '(' expr ')' '{' menu_purpose_specifier '}'
    = 'IF' '(' expr ')' '{' menu_purpose_specifier '}'
    = 'ELSE' '{' menu_purpose_specifier '}'
    = 'SELECT' '(' expr ')' '{' menu_purpose_selection_list '}'

menu_purpose_selection_list:
    = menu_purpose_selection
    = menu_purpose_selection_list menu_purpose_selection

menu_purpose_selection:
    = 'CASE' expr ':' menu_purpose_specifier
    = 'DEFAULT' ':' menu_purpose_specifier

menu_purpose_list:
    = menu_purpose_item
    = menu_purpose_list '&' menu_purpose_item

menu_purpose_item:
    = 'CATALOG'
    = 'DIAGNOSE'
    = 'LOAD_TO_APPLICATION'
    = 'LOAD_TO_DEVICE'
    = 'MENU'
    = 'PROCESS_VALUE'
    = 'SIMULATION'
    = 'TABLE'
    = string

menu_role:
    = 'ROLE' menu_role_specifier

menu_role_specifier:
    = menu_role_list ';'
    = 'IF' '(' expr ')' '{' menu_role_specifier '}'
    = 'IF' '(' expr ')' '{' menu_role_specifier '}'
    = 'ELSE' '{' menu_role_specifier '}'
    = 'SELECT' '(' expr ')' '{' menu_role_selection_list '}'

menu_role_selection_list:
    = menu_role_selection
    = menu_role_selection_list menu_role_selection

menu_role_selection:
    = 'CASE' expr ':' menu_role_specifier
    = 'DEFAULT' ':' menu_role_specifier

menu_role_list:
    = menu_role_item
    = menu_role_list '&' menu_role_item

menu_role_item:
    = string

menu_style:
    = 'STYLE' menu_style_item ';'

menu_style_item:
    = 'DIALOG'
    = 'GROUP'

```

```
= 'MENU'
= 'PAGE'
= 'TABLE'
= 'WINDOW'
= string
```

A.6.24 METHOD

```
method:
    = 'METHOD' Identifier '{' method_attribute_list '}'
    = 'METHOD' Identifier method_parameters '{' method_attribute_list '}'
```

```
method_parameters:
    = '(' method_parameter_list ')'
```

```
method_parameter_list:
    = method_parameter
    = method_parameter_list ',' method_parameter
```

```
method_parameter:
    = method_parameter_type Identifier
    = method_parameter_type '&' Identifier
    = method_parameter_type Identifier '[' ']'
```

```
method_parameter_type:
    = 'char'
    = 'unsigned' 'char'
    = 'short'
    = 'unsigned' 'short'
    = 'int'
    = 'unsigned' 'int'
    = 'long'
    = 'unsigned' 'long'
    = 'float'
    = 'double'
    = 'time_t'
    = 'DD_STRING'
    = 'DD_ITEM'
```

```
method_attribute_list:
    = method_attribute
    = method_attribute_list method_attribute
```

```
method_attribute:
    = method_definition /* M */
    = method_access /* O */
    = variable_class /* O */
    = help /* O */
    = label /* O */
    = method_type /* O */
    = validity /* O */
```

```
method_definition:
    = 'DEFINITION' c_compound_statement
```

```
method_access:
    = 'ACCESS' 'OFFLINE' ';'
    = 'ACCESS' 'ONLINE' ';'

```

```
method_type:
    = 'TYPE' method_return_type
```

```
method_return_type:
    = 'char'
```



```

= 'unsigned' 'char'
= 'int'
= 'unsigned' 'int'
= 'long'
= 'unsigned' 'long'
= 'float'
= 'double'
= 'time_t'
= 'DD_STRING'

```

A.6.25 PROGRAM

```

program:
= 'PROGRAM' Identifier '{' program_attribute_list '}'
= 'PROGRAM' Identifier '{' '}'

program_attribute_list:
= program_attribute
= program_attribute_list program_attribute

program_attribute:
= arguments /* 0 */
= response_codes /* 0 */

arguments:
= 'ARGUMENTS' '{' '}'
= 'ARGUMENTS' '{' arguments_specifier_list '}'

arguments_specifier_list:
= arguments_specifier
= arguments_specifier_list arguments_specifier

arguments_specifier:
= argument_list
= 'IF' '(' expr ')' '{' arguments_specifier_list '}'
= 'IF' '(' expr ')' '{' arguments_specifier_list '}'
= 'ELSE' '{' arguments_specifier_list '}'
= 'SELECT' '(' expr ')' '{' arguments_selection_list '}'

arguments_selection_list:
= arguments_selection
= arguments_selection_list arguments_selection

arguments_selection:
= 'CASE' expr ':' arguments_specifier_list
= 'DEFAULT' ':' arguments_specifier_list

argument_list:
= argument
= argument_list ',' argument

argument:
= Integer
= variable_reference

```

A.6.26 RECORD

```

record:
= 'RECORD' Identifier '{' record_attribute_list '}'

record_attribute_list:
= record_attribute
= record_attribute_list record_attribute

record_attribute:

```

```

= label                /* M */
= members              /* M */
= help                 /* O */
= response_codes       /* O */
= validity             /* O */

```

A.6.27 REFERENCE_ARRAY

Reference_Array has two available keywords, "ITEM_ARRAY" (first option) and "ARRAY" (second option) which may be selected by a profile. The keyword ARRAY shall not be selected if the value_array is used.

NOTE If an EDD application supports more than one profile, the keyword ARRAY may be also used instead of ITEM_ARRAY. In this case, the semantical selection is done by attributes.

```

item_array:
  = 'ITEM_ARRAY' 'OF' item_type Identifier
    '{' item_array_attribute_list '}'
  = 'ARRAY' 'OF' item_type Identifier
    '{' item_array_attribute_list '}'

item_array_attribute_list:
  = item_array_attribute
  = item_array_attribute_list item_array_attribute

item_array_attribute:
  = elements                /* M */
  = help                   /* O */
  = label                  /* O */
  = validity               /* O */

elements:
  = 'ELEMENTS' '{' elements_specifier_list '}'

elements_specifier_list:
  = elements_specifier
  = elements_specifier_list elements_specifier

elements_specifier:
  = element
  = 'IF' '(' expr ')' '{' elements_specifier_list '}'
  = 'IF' '(' expr ')' '{' elements_specifier_list '}'
  = 'ELSE' '{' elements_specifier_list '}'
  = 'SELECT' '(' expr ')' '{' elements_selection_list '}'

elements_selection_list:
  = elements_selection
  = elements_selection_list elements_selection

elements_selection:
  = 'CASE' expr ':' elements_specifier_list
  = 'DEFAULT' ':' elements_specifier_list

element:
  = Integer ',' reference_without_call ';'
  = Integer ',' reference_without_call ',' description_string ';'
  = Integer ',' reference_without_call ',' description_string
    ',' help_string ';'

```

A.6.28 Relations

```

refresh_relation:
  = 'REFRESH' Identifier
    '{' refresh_relation_left ':' refresh_relation_right '}'

```

```

refresh_relation_left:
    = variable_reference_specifier_list

refresh_relation_right:
    = variable_reference_specifier_list

unit_relation:
    = 'UNIT' Identifier
      '{' unit_relation_left ':' unit_relation_right '}'

unit_relation_left:
    = variable_reference_specifier

unit_relation_right:
    = variable_axis_reference_selection_list

wao_relation:
    = 'WRITE_AS_ONE' Identifier '{' wao_specifier '}'

wao_specifier:
    = variable_reference_specifier_list

variable_reference_specifier_list:
    = variable_reference_specifier
    = variable_reference_specifier_list variable_reference_specifier
    = variable_reference_specifier_list ',' variable_reference_specifier

variable_reference_specifier:
    = variable_reference
    = 'IF' '(' expr ')' '{' variable_reference_specifier '}'
    = 'IF' '(' expr ')' '{' variable_reference_specifier '}'
    'ELSE' '{' variable_reference_specifier '}'
    = 'SELECT' '(' expr ')' '{' variable_reference_selection_list '}'

variable_reference_selection_list:
    = variable_reference_selection
    = variable_reference_selection_list variable_reference_selection

variable_reference_selection:
    = 'CASE' expr ':' variable_reference_specifier
    = 'DEFAULT' ':' variable_reference_specifier

variable_axis_reference_selection_list:
    = variable_axis_reference_selection
    = variable_axis_reference_selection_list variable_axis_reference_selection_list

variable_axis_reference_selection:
    = 'CASE' expr ':' variable_axis_reference_specifier
    = 'DEFAULT' ':' variable_axis_reference_specifier

variable_axis_reference_specifier:
    = variable_reference
    = axis_reference
    = 'IF' '(' expr ')' '{' variable_axis_reference_specifier '}'
    = 'IF' '(' expr ')' '{' variable_axis_reference_specifier '}'
    'ELSE' '{' variable_axis_reference_specifier '}'
    = 'SELECT' '(' expr ')' '{' variable_axis_reference_selection_list '}'

```

A.6.29 RESPONSE_CODES

```

response_codes_definition:
    = 'RESPONSE_CODES' Identifier '{' response_codes_attribute_list
      '}'

```

```

response_codes_attribute_list:
    = response_codes_specifier_list

response_codes_specifier_list:
    = response_codes_specifier
    = response_codes_specifier_list response_codes_specifier

response_codes_specifier:
    = response_code
    = 'IF' '(' expr ')' '{' response_codes_specifier_list '}'
    = 'IF' '(' expr ')' '{' response_codes_specifier_list '}'
    'ELSE' '{' response_codes_specifier_list '}'
    = 'SELECT' '(' expr ')' '{' response_codes_selection_list '}'

response_codes_selection_list:
    = response_codes_selection
    = response_codes_selection_list response_codes_selection

response_codes_selection:
    = 'CASE' expr ':' response_codes_specifier_list
    = 'DEFAULT' ':' response_codes_specifier_list

response_code:
    = Integer ',' response_code_type ',' description_string ';'
    = Integer ',' response_code_type ',' description_string ','
      help_string ';'

response_code_type:
    = 'SUCCESS'
    = 'MISC_WARNING'
    = 'DATA_ENTRY_WARNING'
    = 'DATA_ENTRY_ERROR'
    = 'MODE_ERROR'
    = 'PROCESS_ERROR'
    = 'MISC_ERROR'

```

A.6.30 SOURCE

```

source:
    = 'SOURCE' Identifier '{' source_attribute_list '}'

source_attribute_list:
    = source_attribute
    = source_attribute_list source_attribute

source_attribute:
    = members                /* M */
    = emphasis                /* O */
    = source_exit_actions    /* O */
    = help                    /* O */
    = source_init_actions    /* O */
    = label                   /* O */
    = line_color              /* O */
    = line_type               /* O */
    = source_refresh_actions /* O */
    = validity                /* O */
    = source_y_axis           /* O */

source_exit_actions:
    = 'EXIT_ACTIONS' '{' actions_specifier_list '}'

source_init_actions:

```

```

    = 'INIT_ACTIONS' '{' actions_specifier_list '}'

source_refresh_actions:
    = 'REFRESH_ACTIONS' '{' actions_specifier_list '}'

source_y_axis:
    = 'Y_AXIS' axis_specifier

```

A.6.31 TEMPLATE

```

template:
    = 'TEMPLATE' Identifier '{' template_attribute_list '}'

template_attribute_list:
    = template_attribute
    = template_attribute_list template_attribute

template_attribute:
    = template_default_values          /* M */
    = label                          /* M */
    = help                            /* O */

template_default_values:
    = 'DEFAULT_VALUES' '{' template_default_value_list '}'

template_default_value_list:
    = template_default_value
    = template_default_value_list template_default_value

template_default_value:
    = reference_without_call '=' expr specifier

```

A.6.32 VALUE_ARRAY

```

array:
    = 'ARRAY' Identifier '{' array_attribute_list '}'

array_attribute_list:
    = array_attribute
    = array_attribute_list array_attribute

array_attribute:
    = label                          /* M */
    = array_size                     /* M */
    = array_type                     /* M */
    = help                          /* O */
    = response_codes                 /* O */
    = validity                       /* O */

array_size:
    = 'NUMBER_OF_ELEMENTS' Integer ';'

array_type:
    = 'TYPE' Identifier ';'

```

A.6.33 VARIABLE

```

variable:
    = 'VARIABLE' Identifier '{' variable_attribute_list '}'

variable_attribute_list:
    = variable_attribute
    = variable_attribute_list variable_attribute

variable_attribute:
    = variable_class                 /* M */

```

```

= type                                /* M */
= constant_unit                       /* O */
= default_value                       /* O */
= handling                            /* O */
= help                                /* O */
= initial_value                       /* O */
= label                               /* O */
= post_edit_actions                   /* O */
= post_read_actions                   /* O */
= post_write_actions                  /* O */
= pre_edit_actions                    /* O */
= pre_read_actions                    /* O */
= pre_write_actions                   /* O */
= read_timeout                        /* O */
= refresh_actions                     /* O */
= response_codes                      /* O */
= variable_style                      /* O */
= validity                            /* O */
= write_timeout                       /* O */

variable_class:
    = 'CLASS' variable_class_definition ';'

variable_class_definition:
    = variable_class_keyword
    = variable_class_definition '&' variable_class_keyword

variable_class_keyword:
    = 'ALARM'
    = 'ANALOG_INPUT'
    = 'ANALOG_OUTPUT'
    = 'COMPUTATION'
    = 'CONTAINED'
    = 'CORRECTION'
    = 'DEVICE'
    = 'DIAGNOSTIC'
    = 'DIGITAL_INPUT'
    = 'DIGITAL_OUTPUT'
    = 'DISCRETE_INPUT'
    = 'DISCRETE_OUTPUT'
    = 'DYNAMIC'
    = 'FREQUENCY_INPUT'
    = 'FREQUENCY_OUTPUT'
    = 'HART'
    = 'INPUT'
    = 'LOCAL'
    = 'LOCAL_DISPLAY'
    = 'OPERATE'
    = 'OPTIONAL'
    = 'OUTPUT'
    = 'SERVICE'
    = 'TEMPORARY'
    = 'TUNE'

type:
    = 'TYPE' type_specifier

type_specifier:
    = arithmetic_type
    = enumerated_type
    = index_type
    = string_type
    = date_time_type

```

```

    = object_reference

arithmetic_type:
    = 'INTEGER' arithmetic_size arithmetic_options
    = 'UNSIGNED_INTEGER' arithmetic_size arithmetic_options
    = 'DOUBLE' arithmetic_options
    = 'FLOAT' arithmetic_options

arithmetic_size:
    =
    = '(' Integer ')'

arithmetic_options:
    = ';'
    = '{' arithmetic_option_list '}'

arithmetic_option_list:
    = arithmetic_option
    = arithmetic_option_list arithmetic_option

arithmetic_option:
    = default_value
    = initial_value
    = display_format
    = edit_format
    = arithmetic_enumerator_list
    = maximum_value
    = maximum_value_n
    = minimum_value
    = minimum_value_n
    = scaling_factor

display_format:
    = 'DISPLAY_FORMAT' string_specifier

edit_format:
    = 'EDIT_FORMAT' string_specifier

arithmetic_enumerator_list:
    = arithmetic_enumerator
    = arithmetic_enumerator_list ',' arithmetic_enumerator

arithmetic_enumerator:
    = '{' arithmetic_value ',' description_string '}'
    = '{' arithmetic_value ',' description_string ',' help_string '}'

arithmetic_value:
    = Integer
    = Real

scaling_factor:
    = 'SCALING_FACTOR' expr_specifier

maximum_value:
    = 'MAX_VALUE' expr_specifier

maximum_value_n:
    = 'MAX_VALUE1' expr_specifier
    = 'MAX_VALUE2' expr_specifier
    = 'MAX_VALUE3' expr_specifier
    = 'MAX_VALUE4' expr_specifier
    = 'MAX_VALUE5' expr_specifier
    = 'MAX_VALUE6' expr_specifier

```

```

= 'MAX_VALUE7' expr_specifier
= 'MAX_VALUE8' expr_specifier
= 'MAX_VALUE9' expr_specifier

minimum_value:
= 'MIN_VALUE' expr_specifier

minimum_value_n:
= 'MIN_VALUE1' expr_specifier
= 'MIN_VALUE2' expr_specifier
= 'MIN_VALUE3' expr_specifier
= 'MIN_VALUE4' expr_specifier
= 'MIN_VALUE5' expr_specifier
= 'MIN_VALUE6' expr_specifier
= 'MIN_VALUE7' expr_specifier
= 'MIN_VALUE8' expr_specifier
= 'MIN_VALUE9' expr_specifier

enumerated_type:
= 'ENUMERATED' enumerated_size '{' enumerated_option_list
  '}'
= 'BIT_ENUMERATED' enumerated_size '{'
  bit_enumerated_option_list '}'

enumerated_size:
=
= '(' Integer ')'

enumerated_option_list:
= enumerated_option
= enumerated_option_list enumerated_option

enumerated_option:
= default_value
= initial_value
= enumerators_specifier

enumerators_specifier_list:
= enumerators_specifier
= enumerators_specifier_list enumerators_specifier

enumerators_specifier:
= integral_enumerator_list
= 'IF' '(' expr ')' '{' enumerators_specifier_list '}'
= 'IF' '(' expr ')' '{' enumerators_specifier_list '}'
  'ELSE' '{' enumerators_specifier_list '}'
= 'SELECT' '(' expr ')' '{' enumerators_selection_list '}'

enumerators_selection_list:
= enumerators_selection
= enumerators_selection_list enumerators_selection

enumerators_selection:
= 'CASE' expr ':' enumerators_specifier_list
= 'DEFAULT' ':' enumerators_specifier_list

integral_enumerator_list:
= integral_enumerator
= integral_enumerator_list ',' integral_enumerator

integral_enumerator:
= '{' Integer ',' description_string '}'
= '{' Integer ',' description_string ',' help_string '}'

```



```

description_string:
    = string

help_string:
    = string

bit_enumerated_option_list:
    = bit_enumerated_option
    = bit_enumerated_option_list bit_enumerated_option

bit_enumerated_option:
    = default_value
    = initial_value
    = bit_enumerators_specifier

bit_enumerators_specifier_list:
    = bit_enumerators_specifier
    = bit_enumerators_specifier_list bit_enumerators_specifier

bit_enumerators_specifier:
    = bit_enumerator_list
    = 'IF' '(' expr ')' '{' bit_enumerators_specifier_list '}'
    = 'IF' '(' expr ')' '{' bit_enumerators_specifier_list '}'
    = 'ELSE' '{' bit_enumerators_specifier_list '}'
    = 'SELECT' '(' expr ')' '{' bit_enumerators_selection_list '}'

bit_enumerators_selection_list:
    = bit_enumerators_selection
    = bit_enumerators_selection_list bit_enumerators_selection

bit_enumerators_selection:
    = 'CASE' expr ':' bit_enumerators_specifier_list
    = 'DEFAULT' ':' bit_enumerators_specifier_list

bit_enumerator_list:
    = bit_enumerator
    = bit_enumerator_list ',' bit_enumerator

bit_enumerator:
    = '{' Integer ',' string '}'
    = '{' Integer ',' string ',' help_string '}'
    = '{' Integer ',' string ',' variable_class_definition '}'
    = '{' Integer ',' string ',' status_class '}'
    = '{' Integer ',' string ',' string ',' variable_class_definition '}'
    = '{' Integer ',' string ',' string ',' status_class '}'
    = '{' Integer ',' string ',' string ',' method_reference '}'
    = '{' Integer ',' string ',' variable_class_definition ','
        status_class '}'
    = '{' Integer ',' string ',' variable_class_definition ','
        method_reference '}'
    = '{' Integer ',' string ',' status_class ',' method_reference '}'
    = '{' Integer ',' string ',' string ',' variable_class_definition ','
        status_class '}'
    = '{' Integer ',' string ',' string ',' variable_class_definition ','
        method_reference '}'
    = '{' Integer ',' string ',' string ',' status_class ','
        method_reference '}'
    = '{' Integer ',' string ',' variable_class_definition ',' status_class
        ',' method_reference '}'
    = '{' Integer ',' string ',' string ',' variable_class_definition ','
        status_class ',' method_reference '}'

```

```

status_class:
    = status_class_definition

status_class_definition:
    = status_class_keyword
    = status_class_definition '&' status_class_keyword

status_class_keyword:
    = 'HARDWARE'
    = 'SOFTWARE'
    = 'PROCESS'
    = 'MODE'
    = 'DATA'
    = 'MISC'
    = 'EVENT'
    = 'STATE'
    = 'SELF_CORRECTING'
    = 'CORRECTABLE'
    = 'UNCORRECTABLE'
    = 'SUMMARY'
    = 'DETAIL'
    = 'MORE'
    = 'COMM'
    = 'IGNORE_IN_TEMPORARY_MASTER'
    = 'BAD'
    = 'INFO'
    = 'WARNING'
    = 'IGNORE_IN_HOST'

index_type:
    = 'INDEX' item_array_reference ';'
    = 'INDEX' '(' Integer ')' item_array_reference ';'

string_type:
    = 'ASCII' string_size string_options
    = 'BITSTRING' string_size string_options
    = 'EUC' string_size string_options
    = 'PACKED_ASCII' string_size string_options
    = 'PASSWORD' string_size string_options
    = 'VISIBLE' string_size string_options
    = 'OCTET' string_size string_options

string_size:
    = '(' Integer ')'

string_options:
    = ';'
    = '{' string_option_list '}'

string_option_list:
    = string_option
    = string_option_list string_option

string_option:
    = string_default_value
    = string_initial_value
    = string_enumerator_list

string_enumerator_list:
    = string_enumerator
    = string_enumerator_list ',' string_enumerator

string_enumerator:

```

```

    = '{' string ',' description_string '}'
    = '{' string ',' description_string ',' help_string '}'

string_default_value:
    = 'DEFAULT_VALUE' string_specifier

string_initial_value:
    = 'INITIAL_VALUE' string ';'

date_time_type:
    = 'DATE' date_time_options
    = 'DATE_AND_TIME' date_time_options
    = 'DURATION' date_time_options
    = 'TIME' date_time_options
    = 'TIME_VALUE' date_time_size time_value_options

date_time_size:
    =
    = '(' Integer ')'

date_time_options:
    = ';'
    = '{' date_time_option_list '}'

date_time_option_list:
    = date_time_option
    = date_time_option_list date_time_option

date_time_option:
    = default_value
    = initial_value

time_value_options:
    = ';'
    = '{' time_value_option_list '}'

time_value_option_list:
    = time_value_option
    = time_value_option_list time_value_option

time_value_options:
    = default_value
    = display_format
    = edit_format
    = initial_value
    = time_format
    = time_scale

time_format:
    = 'TIME_FORMAT' string_specifier

time_scale:
    = 'TIME_SCALE' time_scale_specifier

time_scale_specifier:
    = time_scale_keyword ';'
    = 'IF' '(' expr ')' '{' time_scale_specifier '}'
    = 'IF' '(' expr ')' '{' time_scale_specifier '}'
    = 'ELSE' '{' time_scale_specifier '}'
    = 'SELECT' '(' expr ')' '{' time_scale_selection_list '}'

time_scale_selection_list:
    = time_scale_selection

```

```

    = time_scale_selection_list time_scale_selection

time_scale_selection:
    = 'CASE' expr ':' time_scale_specifier
    = 'DEFAULT' ':' time_scale_specifier

time_scale_keyword:
    = 'SECONDS'
    = 'MINUTES'
    = 'HOURS'

object_reference:
    = 'OBJECT_REFERENCE' object_reference_value ';'

object_reference_value:
    = 'ROOT'
    = 'PARENT'
    = 'CHILD'
    = 'SELF'
    = 'NEXT'
    = 'PREV'
    = 'FIRST'
    = 'LAST'

constant_unit:
    = 'CONSTANT_UNIT' string_specifier

default_value:
    = 'DEFAULT_VALUE' expr_specifier

initial_value:
    = 'INITIAL_VALUE' expr ';'

pre_edit_actions:
    = 'PRE_EDIT_ACTIONS' '{' actions_specifier_list '}'

post_edit_actions:
    = 'POST_EDIT_ACTIONS' '{' actions_specifier_list '}'

pre_read_actions:
    = 'PRE_READ_ACTIONS' '{' actions_specifier_list '}'

post_read_actions:
    = 'POST_READ_ACTIONS' '{' actions_specifier_list '}'

pre_write_actions:
    = 'PRE_WRITE_ACTIONS' '{' actions_specifier_list '}'

post_write_actions:
    = 'POST_WRITE_ACTIONS' '{' actions_specifier_list '}'

read_timeout:
    = 'READ_TIMEOUT' expr_specifier

refresh_actions:
    = 'REFRESH_ACTIONS' '{' actions_specifier_list '}'

actions_specifier_list:
    = method_specification
    = actions_specifier_list ',' method_specification

method_specification:
    = method_reference

```

```

    = method_definition

variable_style:
    = 'STYLE' string ';'

write_timeout:
    = 'WRITE_TIMEOUT' expr_specifier

expr_specifier:
    = expr ';'
    = 'IF' '(' expr ')' '{' expr_specifier '}'
    = 'IF' '(' expr ')' '{' expr_specifier '}'
    = 'ELSE' '{' expr_specifier '}'
    = 'SELECT' '(' expr ')' '{' expr_selection_list '}'

expr_selection_list:
    = expr_selection
    = expr_selection_list expr_selection

expr_selection:
    = 'CASE' expr ':' expr_specifier
    = 'DEFAULT' ':' expr_specifier

```

A.6.34 VARIABLE_LIST

```

variable_list:
    = 'VARIABLE_LIST' Identifier '{' variable_list_attribute_list '}'

variable_list_attribute_list:
    = variable_list_attribute
    = variable_list_attribute_list variable_list_attribute

variable_list_attribute:
    = members /* M */
    = help /* O */
    = label /* O */
    = response_codes /* O */

```

A.6.35 WAVEFORM

```

waveform:
    = 'WAVEFORM' Identifier '{' waveform_attribute_list '}'

waveform_attribute_list:
    = waveform_attribute
    = waveform_attribute_list waveform_attribute

waveform_attribute:
    = waveform_type /* M */
    = emphasis /* O */
    = waveform_exit_actions /* O */
    = handling /* O */
    = help /* O */
    = waveform_init_actions /* O */
    = waveform_key_points /* O */
    = label /* O */
    = line_color /* O */
    = line_type /* O */
    = waveform_refresh_actions /* O */
    = validity /* O */
    = waveform_y_axis /* O */

```

```

waveform_type:
    = 'TYPE' waveform_type_specifier

```

```

waveform_type_specifier:
    = waveform_yt_type
    = waveform_xy_type
    = waveform_horizontal_type
    = waveform_vertical_type

waveform_yt_type:
    = 'YT' '{' waveform_yt_type_option_list '}'

waveform_yt_type_option_list:
    = waveform_yt_type_option
    = waveform_yt_type_option_list waveform_yt_type_option

waveform_yt_type_option:
    = yt_x_initial                /* M */
    = yt_x_increment              /* M */
    = waveform_y_values           /* M */
    = number_of_points           /* O */

number_of_points:
    = 'NUMBER_OF_POINTS' expr_specifier

yt_x_initial:
    = 'X_INITIAL' expr_specifier

yt_x_increment:
    = 'X_INCREMENT' expr_specifier

waveform_y_values:
    = 'Y_VALUES' '{' values_option_list '}'

waveform_x_values:
    = 'X_VALUES' '{' values_option_list '}'

values_option_list:
    = values_option
    = values_option_list ',' values_option

values_option:
    = reference_without_call
    = Integer
    = Real

waveform_xy_type:
    = 'XY' '{' waveform_xy_type_option_list '}'

waveform_xy_type_option_list:
    = waveform_xy_type_option
    = waveform_xy_type_option_list waveform_xy_type_option

waveform_xy_type_option:
    = waveform_x_values          /* M */
    = waveform_y_values          /* M */
    = number_of_points           /* O */

waveform_horizontal_type:
    = 'HORIZONTAL' '{' waveform_y_values '}'

waveform_vertical_type:
    = 'VERTICAL' '{' waveform_x_values '}'

waveform_y_axis:

```

```

    = 'Y_AXIS' axis_specifier

axis_specifier:
    = axis_reference ';'
    = 'IF' '(' expr ')' '{' axis_specifier '}'
    = 'IF' '(' expr ')' '{' axis_specifier '}'
      'ELSE' '{' axis_specifier '}'
    = 'SELECT' '(' expr ')' '{' axis_selection_list '}'

axis_selection_list:
    = axis_selection
    = axis_selection_list axis_selection

axis_selection:
    = 'CASE' expr ':' axis_specifier
    = 'DEFAULT' ':' axis_specifier

waveform_exit_actions:
    = 'EXIT_ACTIONS' '{' actions_specifier_list '}'

waveform_init_actions:
    = 'INIT_ACTIONS' '{' actions_specifier_list '}'

waveform_key_points:
    = 'KEY_POINTS' '{' key_points_options_list '}'

key_points_options_list:
    = key_points_options
    = key_points_options_list key_points_options

key_points_options:
    = waveform_x_values /* M */
    = waveform_y_values /* M */

waveform_refresh_actions:
    = 'REFRESH_ACTIONS' '{' actions_specifier_list '}'

```

A.6.36 Common attributes

```

emphasis:
    = 'EMPHASIS' boolean_specifier

boolean_specifier:
    = boolean ';'
    = 'IF' '(' expr ')' '{' boolean_specifier '}'
    = 'IF' '(' expr ')' '{' boolean_specifier '}'
      'ELSE' '{' boolean_specifier '}'
    = 'SELECT' '(' expr ')' '{' boolean_selection_list '}'

boolean_selection_list:
    = boolean_selection
    = boolean_selection_list boolean_selection

boolean_selection:
    = 'CASE' expr ':' boolean_specifier
    = 'DEFAULT' ':' boolean_specifier

boolean:
    = 'FALSE'
    = 'TRUE'

handling:
    = 'HANDLING' handling_specifier

```

```

handling_specifier:
    = handling_definition ';'
    = 'IF' '(' expr ')' '{' handling_specifier '}'
    = 'IF' '(' expr ')' '{' handling_specifier '}'
    = 'ELSE' '{' handling_specifier '}'
    = 'SELECT' '(' expr ')' '{' handling_selection_list '}'

handling_selection_list:
    = handling_selection
    = handling_selection_list handling_selection

handling_selection:
    = 'CASE' expr ':' handling_specifier
    = 'DEFAULT' ':' handling_specifier

handling_definition:
    = handling_keyword
    = handling_definition '&' handling_keyword

handling_keyword:
    = 'READ'
    = 'WRITE'

height:
    = 'HEIGHT' display_size_keyword ';'

display_size_keyword:
    = 'XX_SMALL'
    = 'X_SMALL'
    = 'SMALL'
    = 'MEDIUM'
    = 'XX_LARGE'
    = 'X_LARGE'
    = 'LARGE'

help:
    = 'HELP' string_specifier

string_specifier:
    = string ';'
    = 'IF' '(' expr ')' '{' string_specifier '}'
    = 'IF' '(' expr ')' '{' string_specifier '}'
    = 'ELSE' '{' string_specifier '}'
    = 'SELECT' '(' expr ')' '{' string_selection_list '}'

string_selection_list:
    = string_selection
    = string_selection_list string_selection

string_selection:
    = 'CASE' expr ':' string_specifier
    = 'DEFAULT' ':' string_specifier

string:
    = string_list

string_list:
    = string_terminal
    = string_list '+' string_terminal

string_terminal:
    = string_literal
    = variable_reference

```



```

    = variable_reference '(' Integer ')'
    = dictionary_reference

string_literal:
    = String
    = string_literal String

label:
    = 'LABEL' string_specifier

line_color:
    = 'LINE_COLOR' line_color_specifier

line_color_specifier:
    = color ';'
    = 'IF' '(' expr ')' '{' line_color_specifier '}'
    = 'IF' '(' expr ')' '{' line_color_specifier '}'
    = 'ELSE' '{' line_color_specifier '}'
    = 'SELECT' '(' expr ')' '{' line_color_selection_list '}'

line_color_selection_list:
    = line_color_selection
    = line_color_selection_list line_color_selection

line_color_selection:
    = 'CASE' expr ':' line_color_specifier
    = 'DEFAULT' ':' line_color_specifier

color:
    = Integer
    = variable_reference

line_type:
    = 'LINE_TYPE' line_type_specifier

line_type_specifier:
    = line_type_keyword ';'
    = 'IF' '(' expr ')' '{' line_type_specifier '}'
    = 'IF' '(' expr ')' '{' line_type_specifier '}'
    = 'ELSE' '{' line_type_specifier '}'
    = 'SELECT' '(' expr ')' '{' line_type_selection_list '}'

line_type_selection_list:
    = line_type_selection
    = line_type_selection_list line_type_selection

line_type_selection:
    = 'CASE' expr ':' line_type_specifier
    = 'DEFAULT' ':' line_type_specifier

line_type_keyword:
    = 'DATA'
    = 'DATA1'
    = 'DATA2'
    = 'DATA3'
    = 'DATA4'
    = 'DATA5'
    = 'DATA6'
    = 'DATA7'
    = 'DATA8'
    = 'DATA9'
    = 'LOW_LOW_LIMIT'

```

```

= 'LOW_LIMIT'
= 'HIGH_LIMIT'
= 'HIGH_HIGH_LIMIT'
= 'TRANSPARENT'

members:
= 'MEMBERS' '{' members_specifier_list '}'

members_specifier_list:
= members_specifier
= members_specifier_list members_specifier

members_specifier:
= member
= 'IF' '(' expr ')' '{' members_specifier_list '}'
= 'IF' '(' expr ')' '{' members_specifier_list '}'
  'ELSE' '{' members_specifier_list '}'
= 'SELECT' '(' expr ')' '{' members_selection_list '}'

members_selection_list:
= members_selection
= members_selection_list members_selection

members_selection:
= 'CASE' expr ':' members_specifier_list
= 'DEFAULT' ':' members_specifier_list

member:
= Identifier ',' reference_without_call ','
= Identifier ',' reference_without_call ',' string ';'
= Identifier ',' reference_without_call ',' string ',' string ';'

response_codes:
= 'RESPONSE_CODES' '{' response_codes_specifier_list '}'
= 'RESPONSE_CODES' response_code_reference_specifier

response_code_reference_specifier:
= response_code_reference
= 'IF' '(' expr ')' '{' response_code_reference_specifier '}'
= 'IF' '(' expr ')' '{' response_code_reference_specifier
  '}'
  'ELSE' '{' response_code_reference_specifier '}'
= 'SELECT' '(' expr ')' '{'
  response_code_reference_selection_list '}'

response_code_reference_selection_list:
= response_code_reference_selection
= response_code_reference_selection_list
  response_code_reference_selection

response_code_reference_selection:
= 'CASE' expr ':' response_code_reference_specifier
= 'DEFAULT' ':' response_code_reference_specifier

validity:
= 'VALIDITY' boolean_specifier

width:
= 'WIDTH' display_size_keyword ';'

```

A.6.37 Expression

primary_expr:

```
= Integer
= Real
= String
= variable_reference
= 'ARRAY_INDEX'
= 'CURRENT_ROLE'
= 'PI'
= '(' expr ')'
```

postfix_expr:

```
= primary_expr
= Identifier '(' ')'
= Identifier '(' c_argument_expr_list ')'
= axis_reference '.' axis_attribute_selector
= list_reference '.' list_attribute_selector
= variable_reference '.' variable_attribute_selector
```

axis_attribute_selector:

```
= 'MAX_VALUE'
= 'MIN_VALUE'
= 'SCALING'
= 'VIEW_MAX'
= 'VIEW_MIN'
```

list_attribute_selector:

```
= 'CAPACITY'
= 'COUNT'
```

variable_attribute_selector:

```
= 'DEFAULT_VALUE'
= 'INITIAL_VALUE'
= 'MAX_VALUE'
= 'MIN_VALUE'
= 'SCALING_FACTOR'
= 'VARIABLE_STATUS'
```

unary_expr:

```
= postfix_expr
= unary_operator unary_expr
```

unary_operator:

```
= '+'
= '-'
= '~'
= '!'
= '!'
```

multiplicative_expr:

```
= unary_expr
= multiplicative_expr '*' unary_expr
= multiplicative_expr '/' unary_expr
= multiplicative_expr '%' unary_expr
```

additive_expr:

```
= multiplicative_expr
= additive_expr '+' multiplicative_expr
= additive_expr '-' multiplicative_expr
```

shift_expr:

```
= additive_expr
= shift_expr '<<' additive_expr
= shift_expr '>>' additive_expr
```

relational_expr:

```

= shift_expr
= relational_expr '<' shift_expr
= relational_expr '>' shift_expr
= relational_expr '<=' shift_expr
= relational_expr '>=' shift_expr

equality_expr:
= relational_expr
= equality_expr '==' relational_expr
= equality_expr '!=' relational_expr

and_expr:
= equality_expr
= and_expr '&' equality_expr

exclusive_or_expr:
= and_expr
= exclusive_or_expr '^' and_expr

inclusive_or_expr:
= exclusive_or_expr
= inclusive_or_expr '|' exclusive_or_expr

logical_and_expr:
= inclusive_or_expr
= logical_and_expr '&&' inclusive_or_expr

logical_or_expr:
= logical_and_expr
= logical_or_expr '||' logical_and_expr

conditional_expr:
= logical_or_expr
= logical_or_expr '?' expr ':' conditional_expr

expr:
= conditional_expr

```

A.6.38 C-Grammar

```

c_primary_expr:
= c_char
= c_dictionary
= c_identifier
= c_integer
= c_real
= c_string
= c_reference
= '(' c_expr ')'

c_char:
= Character

c_dictionary:
= '[' Identifier ']'

c_identifier:
= Identifier

c_integer:
= Integer

c_real:
= Real

```

```

c_string:
    = c_string_literal

c_string_literal:
    = string_literal

c_reference:
    = reference

c_postfix_expr:
    = c_primary_expr
    = c_postfix_expr '[' c_expr ']'
    = Identifier '(' ')'
    = Identifier '(' c_argument_expr_list ')'
    = c_postfix_expr '.' Identifier
    = c_postfix_expr '.' selector
    = c_postfix_expr '++'
    = c_postfix_expr '--'

c_argument_expr_list:
    = c_assignment_expr
    = c_argument_expr_list ',' c_assignment_expr

c_unary_expr:
    = c_postfix_expr
    = '++' c_unary_expr
    = '--' c_unary_expr
    = c_unary_operator c_postfix_expr

c_unary_operator:
    = '+'
    = '-'
    = '~'
    = '!'

c_multiplicative_expr:
    = c_unary_expr
    = c_multiplicative_expr '*' c_unary_expr
    = c_multiplicative_expr '/' c_unary_expr
    = c_multiplicative_expr '%' c_unary_expr

c_additive_expr:
    = c_multiplicative_expr
    = c_additive_expr '+' c_multiplicative_expr
    = c_additive_expr '-' c_multiplicative_expr

c_shift_expr:
    = c_additive_expr
    = c_shift_expr '<<' c_additive_expr
    = c_shift_expr '>>' c_additive_expr

c_relational_expr:
    = c_shift_expr
    = c_relational_expr '<' c_shift_expr
    = c_relational_expr '>' c_shift_expr
    = c_relational_expr '<=' c_shift_expr
    = c_relational_expr '>=' c_shift_expr

c_equality_expr:
    = c_relational_expr
    = c_equality_expr '==' c_relational_expr
    = c_equality_expr '!=' c_relational_expr

```

```

c_and_expr:
    = c_equality_expr
    = c_and_expr '&' c_equality_expr

c_exclusive_or_expr:
    = c_and_expr
    = c_exclusive_or_expr '^' c_and_expr

c_inclusive_or_expr:
    = c_exclusive_or_expr
    = c_inclusive_or_expr '|' c_exclusive_or_expr

c_logical_and_expr:
    = c_inclusive_or_expr
    = c_logical_and_expr '&&' c_inclusive_or_expr

c_logical_or_expr:
    = c_logical_and_expr
    = c_logical_or_expr '||' c_logical_and_expr

c_conditional_expr:
    = c_logical_or_expr
    = c_logical_or_expr '?' c_logical_or_expr ':' c_conditional_expr

c_assignment_expr:
    = c_conditional_expr
    = c_unary_expr c_assignment_operator c_assignment_expr

c_assignment_operator:
    = '='
    = '*='
    = '/='
    = '%='
    = '+='
    = '-='
    = '>>='
    = '<<='
    = '&='
    = '^='
    = '|='

c_expr:
    = c_assignment_expr
    = c_expr ',' c_assignment_expr

c_constant_expr:
    = c_conditional_expr

c_declaration:
    = c_declaration_specifiers c_declarator_list ';'

c_declaration_specifiers:
    = c_type_specifier
    = c_declaration_specifiers c_type_specifier

c_type_specifier:
    = 'char'
    = 'short'
    = 'int'
    = 'long'
    = 'signed'
    = 'unsigned'

```

```

    = 'float'
    = 'double'
    = 'time_t'
    = 'DD_STRING'

c_declarator_list:
    = c_declarator
    = c_declarator_list ',' c_declarator

c_declarator:
    = Identifier
    = Identifier c_declarator_array_specifier_list

c_declarator_array_specifier_list:
    = c_declarator_array_specifier
    = c_declarator_array_specifier_list c_declarator_array_specifier

c_declarator_array_specifier:
    = '[' ']'
    = '[' c_constant_expr ']'

c_statement:
    = c_labeled_statement
    = c_compound_statement
    = c_expr_statement
    = c_selection_statement
    = c_iteration_statement
    = c_jump_statement

c_labeled_statement:
    = 'case' c_constant_expr ':' c_statement
    = 'default' ':' c_statement

c_compound_statement:
    = '{' '}'
    = '{' c_statement_list '}'
    = '{' c_declaration_list '}'
    = '{' c_declaration_list c_statement_list '}'

c_declaration_list:
    = c_declaration
    = c_declaration_list c_declaration

c_statement_list:
    = c_statement
    = c_statement_list c_statement

c_expr_statement:
    = ';'
    = c_expr ';'

c_selection_statement:
    = 'if' '(' c_expr ')' c_statement
    = 'if' '(' c_expr ')' c_statement 'else' c_statement
    = 'switch' '(' c_expr ')' c_statement

c_iteration_statement:
    = 'do' c_statement 'while' '(' c_expr ')' ';'
    = 'while' '(' c_expr ')' c_statement
    = 'for' '('
        c_statement
        'for' '('
            ';'
            ';' c_expr ')'
        c_statement
    = 'for' '('
        ';'
        ';' c_expr ')'
        c_statement

```

```

= 'for' '('
  c_statement
= 'for' '('
  c_statement
= 'for' '(' c_expr ';'
  c_statement
= 'for' '(' c_expr ';'
  c_statement
= 'for' '(' c_expr ';' c_expr ';'
  c_statement
= 'for' '(' c_expr ';' c_expr ';' c_expr ';'
  c_statement

```

```

c_jump_statement:
= 'continue' ';'
= 'break' ';'
= 'return' ';'
= 'return' c_expr ';'

```

A.6.39 Redefinition

```

array_redefinition:
= 'DELETE' 'ARRAY' Identifier ';'
= 'REDEFINE' 'ARRAY' Identifier '{' array_attribute_list '}'
= 'ARRAY' Identifier '{' array_attribute_redefinition_list '}'

array_attribute_redefinition_list:
= array_attribute_redefinition
= array_attribute_redefinition_list array_attribute_redefinition

array_attribute_redefinition:
= array_size_redefinition
= array_type_redefinition
= help_redefinition
= required_label_redefinition
= response_codes_reference_redefinition
= validity_redefinition

array_size_redefinition:
= 'REDEFINE' array_size

array_type_redefinition:
= 'REDEFINE' array_type

axis_redefinition:
= 'DELETE' 'AXIS' Identifier ';'
= 'REDEFINE' 'AXIS' Identifier '{' '}'
= 'REDEFINE' 'AXIS' Identifier '{' axis_attribute_list '}'
= 'AXIS' Identifier '{' axis_attribute_redefinition_list '}'

axis_attribute_redefinition_list:
= axis_attribute_redefinition
= axis_attribute_redefinition_list axis_attribute_redefinition

axis_attribute_redefinition:
= constant_unit_redefinition
= help_redefinition
= optional_label_redefinition
= maximum_value_redefinition
= minimum_value_redefinition
= axis_scaling_redefinition

axis_scaling_redefinition:

```



```

    = 'DELETE' 'SCALING' ';'
    = 'REDEFINE' axis_scaling

block_redefinition:
    = 'DELETE' 'BLOCK' Identifier ';'
    = 'REDEFINE' 'BLOCK' Identifier '{' '}'
    = 'REDEFINE' 'BLOCK' Identifier '{' block_attribute_list '}'
    = 'BLOCK' Identifier '{' block_attribute_redefinition_list '}'

block_attribute_redefinition_list:
    = block_attribute_redefinition
    = block_attribute_redefinition_list block_attribute_redefinition

block_attribute_redefinition:
    = block_a_characteristics_redefinition
    = required_label_redefinition
    = block_a_parameters_redefinition
    = block_a_axis_items_redefinition
    = block_a_chart_items_redefinition
    = block_a_collection_items_redefinition
    = block_a_edit_display_items_redefinition
    = block_a_file_items_redefinition
    = block_a_graph_items_redefinition
    = block_a_grid_items_redefinition
    = help_redefinition
    = block_a_image_items_redefinition
    = block_a_item_array_items_redefinition
    = block_a_list_items_redefinition
    = block_a_local_parameters_redefinition
    = block_a_menu_items_redefinition
    = block_a_method_items_redefinition
    = block_a_parameter_lists_redefinition
    = block_a_refresh_items_redefinition
    = block_a_source_items_redefinition
    = block_a_unit_items_redefinition
    = block_a_wao_items_redefinition
    = block_a_waveform_items_redefinition
    = block_a_charts_redefinition
    = block_a_lists_redefinition
    = block_a_graphs_redefinition
    = block_a_grids_redefinition
    = block_a_menus_redefinition
    = block_a_methods_redefinition
    = block_b_number_redefinition
    = block_b_type_redefinition

block_a_characteristics_redefinition:
    = 'REDEFINE' block_a_characteristics

block_a_parameters_redefinition:
    = 'PARAMETERS' '{' parameter_redefinition_list '}'

parameter_redefinition_list:
    = parameter_redefinition
    = parameter_redefinition_list parameter_redefinition

parameter_redefinition:
    = 'REDEFINE' member
    = 'ADD' member

block_a_axis_items_redefinition:
    = 'DELETE' 'AXIS_ITEMS' ';'
    = 'REDEFINE' block_a_axis_items

```

```

block_a_chart_items_redefinition:
    = 'DELETE' 'CHART_ITEMS' ';'
    = 'REDEFINE' block_a_chart_items

block_a_collection_items_redefinition:
    = 'DELETE' 'COLLECTION_ITEMS' ';'
    = 'REDEFINE' block_a_collection_items

block_a_edit_display_items_redefinition:
    = 'DELETE' 'EDIT_DISPLAY_ITEMS' ';'
    = 'REDEFINE' block_a_edit_display_items

block_a_file_items_redefinition:
    = 'DELETE' 'FILE_ITEMS' ';'
    = 'REDEFINE' block_a_file_items

block_a_graph_items_redefinition:
    = 'DELETE' 'GRAPH_ITEMS' ';'
    = 'REDEFINE' block_a_graph_items

block_a_grid_items_redefinition:
    = 'DELETE' 'GRID_ITEMS' ';'
    = 'REDEFINE' block_a_grid_items

block_a_image_items_redefinition:
    = 'DELETE' 'IMAGE_ITEMS' ';'
    = 'REDEFINE' block_a_image_items

block_a_item_array_items_redefinition:
    = 'DELETE' 'ITEM_ARRAY_ITEMS' ';'
    = 'REDEFINE' block_a_item_array_items

block_a_list_items_redefinition:
    = 'DELETE' 'LIST_ITEMS' ';'
    = 'REDEFINE' block_a_list_items

block_a_local_parameters_redefinition:
    = 'LOCAL_PARAMETERS' '{' parameter_redefinition_list '}'

block_a_menu_items_redefinition:
    = 'DELETE' 'MENU_ITEMS' ';'
    = 'REDEFINE' block_a_menu_items

block_a_method_items_redefinition:
    = 'DELETE' 'METHOD_ITEMS' ';'
    = 'REDEFINE' block_a_method_items

block_a_parameter_lists_redefinition:
    = 'PARAMETER_LISTS' '{' parameter_redefinition_list '}'

block_a_refresh_items_redefinition:
    = 'DELETE' 'REFRESH_ITEMS' ';'
    = 'REDEFINE' block_a_refresh_items

block_a_source_items_redefinition:
    = 'DELETE' 'SOURCE_ITEMS' ';'
    = 'REDEFINE' block_a_source_items

block_a_unit_items_redefinition:
    = 'DELETE' 'UNIT_ITEMS' ';'
    = 'REDEFINE' block_a_unit_items

```

```
block_a_wao_items_redefinition:
    = 'DELETE' 'WRITE_AS_ONE_ITEMS' ';'
    = 'REDEFINE' block_a_wao_items

block_a_waveform_items_redefinition:
    = 'DELETE' 'WAVEFORM_ITEMS' ';'
    = 'REDEFINE' block_a_waveform_items

block_a_charts_redefinition:
    = 'DELETE' 'CHARTS' ';'
    = 'REDEFINE' block_a_charts
    = 'CHARTS' '{' block_a_member_redefinition_list '}'

block_a_lists_redefinition:
    = 'DELETE' 'LISTS' ';'
    = 'REDEFINE' block_a_lists
    = 'LISTS' '{' block_a_member_redefinition_list '}'

block_a_graphs_redefinition:
    = 'DELETE' 'GRAPHS' ';'
    = 'REDEFINE' block_a_graphs
    = 'GRAPHS' '{' block_a_member_redefinition_list '}'

block_a_grids_redefinition:
    = 'DELETE' 'GRIDS' ';'
    = 'REDEFINE' block_a_grids
    = 'GRIDS' '{' block_a_member_redefinition_list '}'

block_a_menus_redefinition:
    = 'DELETE' 'MENUS' ';'
    = 'REDEFINE' block_a_menus
    = 'MENUS' '{' block_a_member_redefinition_list '}'

block_a_methods_redefinition:
    = 'DELETE' 'METHODS' ';'
    = 'REDEFINE' block_a_methods
    = 'METHODS' '{' block_a_member_redefinition_list '}'

block_a_member_redefinition_list:
    = block_a_member_redefinition
    = block_a_member_redefinition_list block_a_member_redefinition

block_a_member_redefinition:
    = 'DELETE' Identifier ';'
    = 'REDEFINE' member
    = 'ADD' member

block_b_type_redefinition:
    = 'REDEFINE' block_b_type

block_b_number_redefinition:
    = 'REDEFINE' block_b_number

chart_redefinition:
    = 'DELETE' 'CHART' Identifier ';'
    = 'REDEFINE' 'CHART' Identifier '{' chart_attribute_list '}'
    = 'CHART' Identifier '{' chart_attribute_redefinition_list '}'

chart_attribute_redefinition_list:
    = chart_attribute_redefinition
    = chart_attribute_redefinition_list chart_attribute_redefinition

chart_attribute_redefinition:
```

```

= members_redefinition
= chart_cycle_time_redefinition
= height_redefinition
= help_redefinition
= optional_label_redefinition
= chart_length_redefinition
= chart_type_redefinition
= validity_redefinition
= width_redefinition

chart_cycle_time_redefinition:
= 'DELETE' 'CYCLE_TIME' ';'
= 'REDEFINE' chart_cycle_time

chart_length_redefinition:
= 'DELETE' 'LENGTH' ';'
= 'REDEFINE' chart_length

chart_type_redefinition:
= 'DELETE' 'TYPE' ';'
= 'REDEFINE' chart_type

collection_redefinition:
= 'DELETE' 'COLLECTION' Identifier ';'
= 'REDEFINE' 'COLLECTION' Identifier '{' '}'
= 'REDEFINE' 'COLLECTION' 'OF' item_type Identifier '{' '}'
= 'REDEFINE' 'COLLECTION' Identifier '{' collection_attribute_list '}'
= 'REDEFINE' 'COLLECTION' 'OF' item_type Identifier '{'
collection_attribute_list '}'
= 'COLLECTION' Identifier '{'
collection_attribute_redefinition_list '}'
= 'COLLECTION' 'OF' item_type Identifier '{'
collection_attribute_redefinition_list '}'

collection_attribute_redefinition_list:
= collection_attribute_redefinition
= collection_attribute_redefinition_list
collection_attribute_redefinition

collection_attribute_redefinition:
= members_redefinition
= help_redefinition
= optional_label_redefinition
= validity_redefinition

command_redefinition:
= 'DELETE' 'COMMAND' Identifier ';'
= 'REDEFINE' 'COMMAND' Identifier '{' '}'
= 'REDEFINE' 'COMMAND' Identifier '{' command_attribute_list '}'
= 'COMMAND' Identifier '{' command_attribute_redefinition_list '}'

command_attribute_redefinition_list:
= command_attribute_redefinition
= command_attribute_redefinition_list command_attribute_redefinition

command_attribute_redefinition:
= command_transaction_redefinition
= command_index_redefinition
= command_slot_redefinition
= command_sub_slot_redefinition
= command_connection_redefinition
= command_header_redefinition
= command_module_redefinition

```

```
= response_codes_reference_redefinition

command_transaction_redefinition:
= 'ADD' command_transaction
= 'REDEFINE' command_transaction
= 'DELETE' 'TRANSACTION' ';'
= 'DELETE' 'TRANSACTION' Integer ';'

command_index_redefinition:
= 'DELETE' 'INDEX' ';'
= 'REDEFINE' command_index

command_slot_redefinition:
= 'DELETE' 'SLOT' ';'
= 'REDEFINE' command_slot

command_sub_slot_redefinition:
= 'DELETE' 'SUB_SLOT' ';'
= 'REDEFINE' command_sub_slot

command_connection_redefinition:
= 'DELETE' 'CONNECTION' ';'
= 'REDEFINE' command_connection

command_header_redefinition:
= 'DELETE' 'HEADER' ';'
= 'REDEFINE' command_header

command_module_redefinition:
= 'DELETE' 'MODULE' ';'
= 'REDEFINE' command_module

component_redefinition:
= 'DELETE' 'COMPONENT' Identifier ';'
= 'REDEFINE' 'COMPONENT' Identifier '{' component_attribute_list '}'
= 'COMPONENT' Identifier '{' component_attribute_redefinition_list '}'

component_attribute_redefinition_list:
= component_attribute_redefinition
= component_attribute_redefinition_list
component_attribute_redefinition

component_attribute_redefinition:
= required_label_redefinition
= can_delete_redefinition
= check_configuration_redefinition
= classification_redefinition
= component_parent_redefinition
= component_path_redefinition
= component_relations_redefinition
= declaration_redefinition
= detect_method_redefinition
= edd_redefinition
= help_redefinition
= initial_values_redefinition
= protocol_redefinition
= redundancy_redefinition
= scan_method_redefinition
= scan_list_redefinition
= supplied_interface_redefinition

can_delete_redefinition:
= 'DELETE' 'CAN_DELETE' ';;'
```

```

    = 'REDEFINE' can_delete

check_configuration_redefinition:
    = 'DELETE' 'CHECK_CONFIGURATION' ';'
    = 'REDEFINE' check_configuration

classification_redefinition:
    = 'DELETE' 'CLASSIFICATION' ';'
    = 'REDEFINE' classification

component_parent_redefinition:
    = 'DELETE' 'COMPONENT_PARENT' ';'
    = 'REDEFINE' component_parent

component_path_redefinition:
    = 'DELETE' 'COMPONENT_PATH' ';'
    = 'REDEFINE' component_path

component_relations_redefinition:
    = 'DELETE' 'COMPONENT_REDEFINITIONS' ';'
    = 'REDEFINE' component_relations

declaration_redefinition:
    = 'DELETE' 'DECLARATION' ';'
    = 'REDEFINE' declaration

detect_method_redefinition:
    = 'DELETE' 'DETECT' ';'
    = 'REDEFINE' detect_method

edd_redefinition:
    = 'DELETE' 'EDD' ';'
    = 'REDEFINE' edd

initial_values_redefinition:
    = 'DELETE' 'INITIAL_VALUES' ';'
    = 'REDEFINE' initial_values

protocol_redefinition:
    = 'DELETE' 'PROTOCOL' ';'
    = 'REDEFINE' protocol

redundancy_redefinition:
    = 'DELETE' 'REDUNDANCY' ';'
    = 'REDEFINE' redundancy

scan_method_redefinition:
    = 'DELETE' 'SCAN' ';'
    = 'REDEFINE' scan_method

scan_list_redefinition:
    = 'DELETE' 'SCAN_LIST' ';'
    = 'REDEFINE' scan_list

supplied_interface_redefinition:
    = 'DELETE' 'SUPPLIED_INTERFACE' ';'
    = 'REDEFINE' supplied_interface

component_folder_redefinition:
    = 'DELETE' 'COMPONENT_FOLDER' Identifier ';'
    = 'REDEFINE' 'COMPONENT_FOLDER' Identifier '{'
        component_folder_attribute_list '}'
    = 'COMPONENT_FOLDER' Identifier '{'

```

```

    component_folder_attribute_redefinition_list '}'

component_folder_attribute_redefinition_list:
    = component_folder_attribute_redefinition
    = component_folder_attribute_redefinition_list
    component_folder_attribute_redefinition

component_folder_attribute_redefinition:
    = required_label_redefinition
    = classification_redefinition
    = component_parent_redefinition
    = component_path_redefinition
    = help_redefinition
    = protocol_redefinition

component_reference_redefinition:
    = 'DELETE' 'COMPONENT_REFERENCE' Identifier ';'
    = 'REDEFINE' 'COMPONENT_REFERENCE' Identifier '{'
        component_reference_attribute_list '}'
    = 'COMPONENT_REFERENCE' Identifier '{'
        component_reference_attribute_redefinition_list '}'

component_reference_attribute_redefinition_list:
    = component_reference_attribute_redefinition
    = component_reference_attribute_redefinition_list
    component_reference_attribute_redefinition

component_reference_attribute_redefinition:
    = classification_redefinition
    = component_parent_redefinition
    = component_path_redefinition
    = device_revision_attribute_redefinition
    = device_type_attribute_redefinition
    = manufacturer_attribute_redefinition
    = protocol_redefinition

device_revision_attribute_redefinition:
    = 'DELETE' 'DEVICE_REVISION' ';'
    = 'REDEFINE' device_revision_attribute

device_type_attribute_redefinition:
    = 'DELETE' 'DEVICE_TYPE' ';'
    = 'REDEFINE' device_type_attribute

manufacturer_attribute_redefinition:
    = 'DELETE' 'MANUFACTURER' ';'
    = 'REDEFINE' manufacturer_attribute

component_relation_redefinition:
    = 'DELETE' 'COMPONENT_RELATION' Identifier ';'
    = 'REDEFINE' 'COMPONENT_RELATION' Identifier '{'
        component_relation_attribute_list '}'
    = 'COMPONENT_RELATION' Identifier '{'
        component_relation_attribute_redefinition_list '}'

component_relation_attribute_redefinition_list:
    = component_relation_attribute_redefinition
    = component_relation_attribute_redefinition_list
    component_relation_attribute_redefinition

component_relation_attribute_redefinition:
    = components_redefinition
    = required_label_redefinition

```

```

= relation_type_redefinition
= addressing_redefinition
= help_redefinition
= maximum_number_redefinition
= minimum_number_redefinition
= required_interface_redefinition
= supplied_interface_redefinition

components_redefinition:
= 'REDEFINE' components
= 'COMPONENTS' '{' component_relation_reference_redefinition_list '}'

component_relation_reference_redefinition_list:
= component_relation_reference_redefinition
= component_relation_reference_redefinition_list ','
  component_relation_reference_redefinition

component_relation_reference_redefinition:
= 'DELETE' Identifier ';'
= 'REDEFINE' component_relation_reference_specifier
= 'ADD' component_relation_reference_specifier
= Identifier '{' component_relation_arg_redefinition_list '}'

component_relation_arg_redefinition_list:
= component_relation_arg_redefinition
= component_relation_arg_redefinition_list
  component_relation_arg_redefinition

component_relation_arg_redefinition:
= minimum_number_redefinition
= maximum_number_redefinition
= auto_create_redefinition
= filter_redefinition
= required_ranges_redefinition

auto_create_redefinition:
= 'DELETE' 'AUTO_CREATE' ';'
= 'REDEFINE' auto_create

filter_redefinition:
= 'DELETE' 'FILTER' ';'
= 'REDEFINE' filter

required_ranges_redefinition:
= 'DELETE' 'REQUIRED_RANGES' ';'
= 'REDEFINE' required_ranges

relation_type_redefinition:
= 'REDEFINE' relation_type

addressing_redefinition:
= 'DELETE' 'ADDRESSING' ';'
= 'REDEFINE' addressing

maximum_number_redefinition:
= 'DELETE' 'MAXIMUM_NUMBER' ';'
= 'REDEFINE' maximum_number

minimum_number_redefinition:
= 'DELETE' 'MINIMUM_NUMBER' ';'
= 'REDEFINE' minimum_number

required_interface_redefinition:

```



```

= 'DELETE' 'REQUIRED_INTERFACE' ';'
= 'REDEFINE' required_interface

connection_redefinition:
= 'DELETE' 'CONNECTION' Identifier ';'
= 'REDEFINE' 'CONNECTION' Identifier '{' '}'
= 'REDEFINE' 'CONNECTION' Identifier '{'
  connection_attribute_list '}'
= 'CONNECTION' Identifier '{'
  connection_attribute_redefinition_list '}'

connection_attribute_redefinition_list:
= connection_attribute_redefinition
= connection_attribute_redefinition_list
  connection_attribute_redefinition

connection_attribute_redefinition:
= appinstance_redefinition

appinstance_redefinition:
= 'REDEFINE' appinstance

domain_redefinition:
= 'DELETE' 'DOMAIN' Identifier ';'
= 'REDEFINE' 'DOMAIN' Identifier '{' domain_attribute_list '}'
= 'DOMAIN' Identifier '{' domain_attribute_redefinition_list '}'

domain_attribute_redefinition_list:
= domain_attribute_redefinition
= domain_attribute_redefinition_list domain_attribute_redefinition

domain_attribute_redefinition:
= handling_redefinition
= response_codes_reference_redefinition

edit_display_redefinition:
= 'DELETE' 'EDIT_DISPLAY' Identifier ';'
= 'REDEFINE' 'EDIT_DISPLAY' Identifier '{' '}'
= 'REDEFINE' 'EDIT_DISPLAY' Identifier '{'
  edit_display_attribute_list '}'
= 'EDIT_DISPLAY' Identifier '{'
  edit_display_attribute_redefinition_list '}'

edit_display_attribute_redefinition_list:
= edit_display_attribute_redefinition
= edit_display_attribute_redefinition_list
  edit_display_attribute_redefinition

edit_display_attribute_redefinition:
= edit_items_redefinition
= required_label_redefinition
= display_items_redefinition
= post_edit_actions_redefinition
= pre_edit_actions_redefinition

edit_items_redefinition:
= 'REDEFINE' edit_items

display_items_redefinition:
= 'DELETE' 'DISPLAY_ITEMS' ';'
= 'REDEFINE' display_items

file_redefinition:

```

```

= 'DELETE' 'FILE' Identifier ';'
= 'REDEFINE' 'FILE' Identifier '{' file_attribute_list '}'
= 'FILE' Identifier '{' file_attribute_redefinition_list '}'

file_attribute_redefinition_list:
= file_attribute_redefinition
= file_attribute_redefinition_list file_attribute_redefinition

file_attribute_redefinition:
= members_redefinition
= optional_label_redefinition
= help_redefinition

graph_redefinition:
= 'DELETE' 'GRAPH' Identifier ';'
= 'REDEFINE' 'GRAPH' Identifier '{' graph_attribute_list '}'
= 'GRAPH' Identifier '{' graph_attribute_redefinition_list '}'

graph_attribute_redefinition_list:
= graph_attribute_redefinition
= graph_attribute_redefinition_list graph_attribute_redefinition

graph_attribute_redefinition:
= members_redefinition
= graph_cycle_time_redefinition
= help_redefinition
= optional_label_redefinition
= height_redefinition
= validity_redefinition
= width_redefinition
= graph_x_axis_redefinition

graph_cycle_time_redefinition:
= 'DELETE' 'CYCLE_TIME' ';'
= 'REDEFINE' graph_cycle_time

graph_x_axis_redefinition:
= 'DELETE' 'X_AXIS' ';'
= 'REDEFINE' graph_x_axis

grid_redefinitions:
= 'DELETE' 'GRID' Identifier ';'
= 'REDEFINE' 'GRID' Identifier '{' grid_attribute_list '}'
= 'GRID' Identifier '{' grid_attribute_redefinition_list '}'

grid_attribute_redefinition_list:
= grid_attribute_redefinition
= grid_attribute_redefinition_list grid_attribute_redefinition

grid_attribute_redefinition:
= grid_vectors_redefinition
= handling_redefinition
= height_redefinition
= help_redefinition
= optional_label_redefinition
= grid_orientation_redefinition
= validity_redefinition
= width_redefinition

grid_vectors_redefinition:
= 'REDEFINE' grid_vectors

grid_orientation_redefinition:

```

```

    = 'DELETE' 'ORIENTATION' ';'
    = 'REDEFINE' grid_orientation

image_redefinition:
    = 'DELETE' 'IMAGE' Identifier ';'
    = 'REDEFINE' 'IMAGE' Identifier '{' image_attribute_list '}'
    = 'IMAGE' Identifier '{' image_attribute_redefinition_list '}'

image_attribute_redefinition_list:
    = image_attribute_redefinition
    = image_attribute_redefinition_list image_attribute_redefinition

image_attribute_redefinition:
    = image_path_redefinition
    = optional_label_redefinition
    = help_redefinition
    = image_link_redefinition
    = validity_redefinition

image_path_redefinition:
    = 'REDEFINE' image_path

image_link_redefinition:
    = 'DELETE' 'LINK' ';'
    = 'REDEFINE' image_link

interface_redefinition:
    = 'DELETE' 'INTERFACE' Identifier ';'
    = 'REDEFINE' 'INTERFACE' Identifier '{' interface_attribute_list '}'
    = 'INTERFACE' Identifier '{' interface_attribute_redefinition_list '}'

interface_attribute_redefinition_list:
    = interface_attribute_redefinition
    = interface_attribute_redefinition_list
      interface_attribute_redefinition

interface_attribute_redefinition:
    = required_declaration_redefinition
    = required_label_redefinition
    = help_redefinition

required_declaration_redefinition:
    = 'REDEFINE' declaration

item_array_redefinition:
    = 'DELETE' 'ITEM_ARRAY' Identifier ';'
    = 'REDEFINE' 'ITEM_ARRAY' 'OF' item_type Identifier '{' '}'
    = 'REDEFINE' 'ITEM_ARRAY' 'OF' item_type Identifier '{'
      item_array_attribute_list '}'
    = 'ITEM_ARRAY' 'OF' item_type Identifier '{'
      item_array_attribute_redefinition_list '}'

item_array_attribute_redefinition_list:
    = item_array_attribute_redefinition
    = item_array_attribute_redefinition_list
      item_array_attribute_redefinition

item_array_attribute_redefinition:
    = elements_redefinition
    = help_redefinition
    = optional_label_redefinition
    = validity_redefinition

```

```

elements_redefinition:
    = 'ELEMENTS' '{' element_redefinition_list '}'
    = 'REDEFINE' elements

element_redefinition_list:
    = element_redefinition
    = element_redefinition_list element_redefinition

element_redefinition:
    = 'DELETE' Integer ';'
    = 'REDEFINE' element
    = 'ADD' element

list_redefinition:
    = 'DELETE' 'LIST' Identifier ';'
    = 'REDEFINE' 'LIST' Identifier '{' list_attribute_list '}'
    = 'LIST' Identifier '{' list_attribute_redefinition_list '}'

list_attribute_redefinition_list:
    = list_attribute_redefinition
    = list_attribute_redefinition_list list_attribute_redefinition

list_attribute_redefinition:
    = list_type_redefinition
    = list_capacity_redefinition
    = list_count_redefinition
    = help_redefinition
    = optional_label_redefinition
    = validity_redefinition

list_type_redefinition:
    = 'REDEFINE' list_type

list_capacity_redefinition:
    = 'DELETE' 'CAPACITY' ';'
    = 'REDEFINE' list_capacity

list_count_redefinition:
    = 'DELETE' 'COUNT' ';'
    = 'REDEFINE' list_count

menu_redefinition:
    = 'DELETE' 'MENU' Identifier ';'
    = 'REDEFINE' 'MENU' Identifier '{' '}'
    = 'REDEFINE' 'MENU' Identifier '{' menu_attribute_list '}'
    = 'MENU' Identifier '{' menu_attribute_redefinition_list '}'

menu_attribute_redefinition_list:
    = menu_attribute_redefinition
    = menu_attribute_redefinition_list menu_attribute_redefinition

menu_attribute_redefinition:
    = menu_items_redefinition
    = required_label_redefinition
    = menu_access_redefinition
    = help_redefinition
    = post_edit_actions_redefinition
    = post_read_actions_redefinition
    = post_write_actions_redefinition
    = pre_edit_actions_redefinition
    = pre_read_actions_redefinition
    = pre_write_actions_redefinition
    = menu_purpose_redefinition

```

```

    = menu_role_redefinition
    = menu_style_redefinition
    = validity_redefinition

menu_items_redefinition:
    = 'REDEFINE' menu_items

menu_access_redefinition:
    = 'DELETE' 'ACCESS' ';'
    = 'REDEFINE' menu_access

menu_purpose_redefinition:
    = 'DELETE' 'PURPOSE' ';'
    = 'REDEFINE' menu_purpose

menu_role_redefinition:
    = 'DELETE' 'ROLE' ';'
    = 'REDEFINE' menu_role

menu_style_redefinition:
    = 'DELETE' 'STYLE' ';'
    = 'REDEFINE' menu_style

method_redefinition:
    = 'DELETE' 'METHOD' Identifier ';'
    = 'REDEFINE' 'METHOD' Identifier '{' '}'
    = 'REDEFINE' 'METHOD' Identifier '{' method_attribute_list '}'
    = 'METHOD' Identifier '{' method_attribute_redefinition_list '}'

method_attribute_redefinition_list:
    = method_attribute_redefinition
    = method_attribute_redefinition_list method_attribute_redefinition

method_attribute_redefinition:
    = method_definition_redefinition
    = method_access_redefinition
    = variable_class_redefinition
    = help_redefinition
    = optional_label_redefinition
    = validity_redefinition

method_definition_redefinition:
    = 'REDEFINE' method_definition

method_access_redefinition:
    = 'DELETE' 'ACCESS' ';'
    = 'REDEFINE' method_access

program_redefinition:
    = 'DELETE' 'PROGRAM' Identifier ';'
    = 'REDEFINE' 'PROGRAM' Identifier '{' '}'
    = 'REDEFINE' 'PROGRAM' Identifier '{' program_attribute_list '}'
    = 'PROGRAM' Identifier '{' program_attribute_redefinition_list '}'

program_attribute_redefinition_list:
    = program_attribute_redefinition
    = program_attribute_redefinition_list program_attribute_redefinition

program_attribute_redefinition:
    = arguments_redefinition
    = response_codes_reference_redefinition

arguments_redefinition:

```

```

= 'DELETE' 'ARGUMENTS' ';'
= 'REDEFINE' arguments

record_redefinition:
= 'DELETE' 'RECORD' Identifier ';'
= 'REDEFINE' 'RECORD' Identifier '{' record_attribute_list '}'
= 'RECORD' Identifier '{' record_attribute_redefinition_list '}'

record_attribute_redefinition_list:
= record_attribute_redefinition
= record_attribute_redefinition_list record_attribute_redefinition

record_attribute_redefinition:
= required_label_redefinition
= members_redefinition
= help_redefinition
= response_codes_reference_redefinition
= validity_redefinition

refresh_relation_redefinition:
= 'DELETE' 'REFRESH' Identifier ';'
= 'REDEFINE' 'REFRESH' Identifier '{' '}'
= 'REDEFINE' 'REFRESH' Identifier
  '{' refresh_relation_left ':' refresh_relation_right '}'

unit_relation_redefinition:
= 'DELETE' 'UNIT' Identifier ';'
= 'REDEFINE' 'UNIT' Identifier '{' '}'
= 'REDEFINE' 'UNIT' Identifier
  '{' unit_relation_left ':' unit_relation_right '}'

wao_relation_redefinition:
= 'DELETE' 'WRITE_AS_ONE' Identifier ';'
= 'REDEFINE' 'WRITE_AS_ONE' Identifier '{' '}'
= 'REDEFINE' 'WRITE_AS_ONE' Identifier
  '{' wao_specifier '}'

response_codes_definition_redefinition:
= 'DELETE' 'RESPONSE_CODES' Identifier ';'
= 'REDEFINE' 'RESPONSE_CODES' Identifier '{' '}'
= 'REDEFINE' 'RESPONSE_CODES' Identifier '{'
  response_codes_specifier_list '}'
= 'RESPONSE_CODES' Identifier '{' response_code_redefinition_list
  '}'

response_code_redefinition_list:
= response_code_redefinition
= response_code_redefinition_list response_code_redefinition

response_code_redefinition:
= 'DELETE' Integer ';'
= 'REDEFINE' response_code
= 'ADD' response_code

emphasis_redefinition:
= 'DELETE' 'EMPHASIS' ';'
= 'REDEFINE' emphasis

height_redefinition:
= 'DELETE' 'HEIGHT' ';'
= 'REDEFINE' height

help_redefinition:

```

```
= 'DELETE' 'HELP' ';'
= 'REDEFINE' help

required_label_redefinition:
= 'REDEFINE' label

optional_label_redefinition:
= 'DELETE' 'LABEL' ';'
= 'REDEFINE' label

line_color_redefinition:
= 'DELETE' 'LINE_COLOR' ';'
= 'REDEFINE' line_color

line_type_redefinition:
= 'DELETE' 'LINE_TYPE' ';'
= 'REDEFINE' line_type

response_codes_reference_redefinition:
= 'DELETE' 'RESPONSE_CODES' ';'
= 'REDEFINE' response_codes

validity_redefinition:
= 'DELETE' 'VALIDITY' ';'
= 'REDEFINE' validity

width_redefinition:
= 'DELETE' 'WIDTH' ';'
= 'REDEFINE' width

members_redefinition:
= 'MEMBERS' '{' member_redefinition_list '}'
= 'REDEFINE' members

member_redefinition_list:
= member_redefinition
= member_redefinition_list member_redefinition

member_redefinition:
= 'DELETE' Identifier ';'
= 'REDEFINE' member
= 'ADD' member

source_redefinition:
= 'DELETE' 'SOURCE' Identifier ';'
= 'REDEFINE' 'SOURCE' Identifier '{' source_attribute_list '}'
= 'SOURCE' Identifier '{' source_attribute_redefinition_list '}'

source_attribute_redefinition_list:
= source_attribute_redefinition
= source_attribute_redefinition_list source_attribute_redefinition

source_attribute_redefinition:
= members_redefinition
= emphasis_redefinition
= source_exit_actions_redefinition
= help_redefinition
= source_init_actions_redefinition
= optional_label_redefinition
= line_color_redefinition
= line_type_redefinition
= source_refresh_actions_redefinition
= validity_redefinition
```

```

    = source_y_axis_redefinition

source_exit_actions_redefinition:
    = 'DELETE' 'REFRESH_ACTIONS' ';'
    = 'REDEFINE' source_exit_actions

source_init_actions_redefinition:
    = 'DELETE' 'INIT_ACTIONS' ';'
    = 'REDEFINE' source_init_actions

source_refresh_actions_redefinition:
    = 'DELETE' 'REFRESH_ACTIONS' ';'
    = 'REDEFINE' source_refresh_actions

source_y_axis_redefinition:
    = 'DELETE' 'Y_AXIS' ';'
    = 'REDEFINE' waveform_y_axis

template_redefinition:
    = 'DELETE' 'TEMPLATE' Identifier ';'
    = 'REDEFINE' 'TEMPLATE' Identifier '{' template_attribute_list '}'
    = 'TEMPLATE' Identifier '{' template_attribute_redefinition_list '}'

template_attribute_redefinition_list:
    = template_attribute_redefinition
    = template_attribute_redefinition_list template_attribute_redefinition

template_attribute_redefinition:
    = template_default_values_redefinition
    = required_label_redefinition
    = help_redefinition

template_default_values_redefinition:
    = 'REDEFINE' template_default_values

variable_redefinition:
    = 'DELETE' 'VARIABLE' Identifier ';'
    = 'REDEFINE' 'VARIABLE' Identifier '{' variable_attribute_list
    '}'
    = 'VARIABLE' Identifier '{' variable_attribute_redefinition_list
    '}'

variable_attribute_redefinition_list:
    = variable_attribute_redefinition
    = variable_attribute_redefinition_list variable_attribute_redefinition

variable_attribute_redefinition:
    = variable_class_redefinition
    = type_redefinition
    = constant_unit_redefinition
    = default_value_redefinition
    = handling_redefinition
    = help_redefinition
    = initial_value_redefinition
    = optional_label_redefinition
    = post_edit_actions_redefinition
    = post_read_actions_redefinition
    = post_write_actions_redefinition
    = pre_edit_actions_redefinition
    = pre_read_actions_redefinition
    = pre_write_actions_redefinition
    = read_timeout_redefinition
    = refresh_actions_redefinition

```



```

    = response_codes_reference_redefinition
    = variable_style_redefinition
    = validity_redefinition
    = write_timeout_redefinition

variable_class_redefinition:
    = 'REDEFINE' variable_class

type_redefinition:
    = 'REDEFINE' 'TYPE' arithmetic_type
    = 'TYPE' arithmetic_type_redefinition
    = 'REDEFINE' 'TYPE' enumerated_type
    = 'TYPE' enumerated_type_redefinition
    = 'REDEFINE' 'TYPE' string_type
    = 'TYPE' string_type_redefinition
    = 'REDEFINE' 'TYPE' date_time_type
    = 'TYPE' date_time_type_redefinition

arithmetic_type_redefinition:
    = 'INTEGER' arithmetic_size '{'
        arithmetic_option_redefinition_list '}'
    = 'UNSIGNED_INTEGER' arithmetic_size '{'
        arithmetic_option_redefinition_list '}'
    = 'FLOAT' '{' arithmetic_option_redefinition_list '}'
    = 'DOUBLE' '{' arithmetic_option_redefinition_list '}'

arithmetic_option_redefinition_list:
    = arithmetic_option_redefinition
    = arithmetic_option_redefinition_list arithmetic_option_redefinition

arithmetic_option_redefinition:
    = default_value_redefinition
    = initial_value_redefinition
    = display_format_redefinition
    = edit_format_redefinition
    = arithmetic_enumerator_list_redefinition
    = maximum_value_redefinition
    = maximum_value_n_redefinition
    = minimum_value_redefinition
    = minimum_value_n_redefinition
    = scaling_factor_redefinition

display_format_redefinition:
    = 'DELETE' 'DISPLAY_FORMAT' ';'
    = 'REDEFINE' display_format

edit_format_redefinition:
    = 'DELETE' 'EDIT_FORMAT' ';'
    = 'REDEFINE' edit_format

arithmetic_enumerator_list_redefinition:
    = 'DELETE' Integer ';'
    = 'DELETE' Real ';'
    = 'REDEFINE' arithmetic_enumerator
    = 'ADD' arithmetic_enumerator

maximum_value_redefinition:
    = 'DELETE' 'MAX_VALUE' ';'
    = 'REDEFINE' maximum_value

maximum_value_n_redefinition:
    = 'DELETE' 'MAX_VALUE1' ';'
    = 'DELETE' 'MAX_VALUE2' ';'

```

```

= 'DELETE' 'MAX_VALUE3' ';'
= 'DELETE' 'MAX_VALUE4' ';'
= 'DELETE' 'MAX_VALUE5' ';'
= 'DELETE' 'MAX_VALUE6' ';'
= 'DELETE' 'MAX_VALUE7' ';'
= 'DELETE' 'MAX_VALUE8' ';'
= 'DELETE' 'MAX_VALUE9' ';'
= 'REDEFINE' maximum_value_n

minimum_value_redefinition:
= 'DELETE' 'MIN_VALUE' ';'
= 'REDEFINE' minimum_value

minimum_value_n_redefinition:
= 'DELETE' 'MIN_VALUE1' ';'
= 'DELETE' 'MIN_VALUE2' ';'
= 'DELETE' 'MIN_VALUE3' ';'
= 'DELETE' 'MIN_VALUE4' ';'
= 'DELETE' 'MIN_VALUE5' ';'
= 'DELETE' 'MIN_VALUE6' ';'
= 'DELETE' 'MIN_VALUE7' ';'
= 'DELETE' 'MIN_VALUE8' ';'
= 'DELETE' 'MIN_VALUE8' ';'
= 'REDEFINE' minimum_value_n

scaling_factor_redefinition:
= 'DELETE' 'SCALING_FACTOR' ';'
= 'REDEFINE' scaling_factor

enumerated_type_redefinition:
= 'ENUMERATED' enumerated_size '{'
    enumerated_option_redefinition_list '}'
= 'BIT_ENUMERATED' enumerated_size '{'
    bit_enumerated_option_redefinition_list '}'

enumerated_option_redefinition_list:
= enumerated_option_redefinition
= enumerated_option_redefinition_list enumerated_option_redefinition

enumerated_option_redefinition:
= default_value_redefinition
= initial_value_redefinition
= integral_enumerator_redefinition

integral_enumerator_redefinition:
= 'DELETE' Integer ';'
= 'REDEFINE' integral_enumerator
= 'ADD' integral_enumerator

bit_enumerated_option_redefinition_list:
= bit_enumerated_option_redefinition
= bit_enumerated_option_redefinition_list

bit_enumerated_option_redefinition

bit_enumerated_option_redefinition:
= default_value_redefinition
= initial_value_redefinition
= bit_enumerators_redefinition

bit_enumerators_redefinition:
= 'DELETE' Integer ';'
= 'REDEFINE' bit_enumerator
= 'ADD' bit_enumerator

```

```

string_type_redefinition:
    = 'ASCII' string_size '{' string_option_redefinition_list '}'
    = 'BITSTRING' string_size '{' string_option_redefinition_list '}'
    = 'EUC' string_size '{' string_option_redefinition_list '}'
    = 'OCTET' string_size '{' string_option_redefinition_list '}'
    = 'PACKED_ASCII' string_size '{' string_option_redefinition_list '}'
    = 'PASSWORD' string_size '{' string_option_redefinition_list '}'
    = 'VISIBLE' string_size '{' string_option_redefinition_list '}'

string_option_redefinition_list:
    = string_option_redefinition
    = string_option_redefinition_list string_option_redefinition

string_option_redefinition:
    = string_default_value_redefinition
    = string_initial_value_redefinition
    = string_enumerator_redefinition

string_default_value_redefinition:
    = 'DELETE' 'DEFAULT_VALUE' ';'
    = 'REDEFINE' string_default_value

string_initial_value_redefinition:
    = 'DELETE' 'INITIAL_VALUE' ';'
    = 'REDEFINE' string_initial_value

string_enumerator_redefinition:
    = 'DELETE' string ';'
    = 'REDEFINE' string_enumerator
    = 'ADD' string_enumerator

date_time_type_redefinition:
    = 'DATE' '{' date_time_option_redefinition_list '}'
    = 'DATE_AND_TIME' '{' date_time_option_redefinition_list '}'
    = 'DURATION' '{' date_time_option_redefinition_list '}'
    = 'TIME' '{' date_time_option_redefinition_list '}'
    = 'TIME_VALUE' date_time_size '{'
        = time_value_option_redefinition_list '}'

date_time_option_redefinition_list:
    = date_time_option_redefinition
    = date_time_option_redefinition_list date_time_option_redefinition

date_time_option_redefinition:
    = default_value_redefinition
    = initial_value_redefinition

time_value_option_redefinition_list:
    = time_value_option_redefinition
    = time_value_option_redefinition_list time_value_option_redefinition

time_value_option_redefinition:
    = default_value_redefinition
    = display_format_redefinition
    = edit_format_redefinition
    = initial_value_redefinition
    = time_format_redefinition
    = time_scale_redefinition

time_format_redefinition:
    = 'DELETE' 'TIME_FORMAT' ';'
    = 'REDEFINE' time_format

```

```

time_scale_redefinition:
    = 'DELETE' 'TIME_SCALE' ';'
    = 'REDEFINE' time_scale

constant_unit_redefinition:
    = 'DELETE' 'CONSTANT_UNIT' ';'
    = 'REDEFINE' constant_unit

default_value_redefinition:
    = 'DELETE' 'DEFAULT_VALUE' ';'
    = 'REDEFINE' default_value

handling_redefinition:
    = 'DELETE' 'HANDLING' ';'
    = 'REDEFINE' handling

initial_value_redefinition:
    = 'DELETE' 'INITIAL_VALUE' ';'
    = 'REDEFINE' initial_value

post_edit_actions_redefinition:
    = 'DELETE' 'POST_EDIT_ACTIONS' ';'
    = 'REDEFINE' post_edit_actions

post_read_actions_redefinition:
    = 'DELETE' 'POST_READ_ACTIONS' ';'
    = 'REDEFINE' post_read_actions

post_write_actions_redefinition:
    = 'DELETE' 'POST_WRITE_ACTIONS' ';'
    = 'REDEFINE' post_write_actions

pre_edit_actions_redefinition:
    = 'DELETE' 'PRE_EDIT_ACTIONS' ';'
    = 'REDEFINE' pre_edit_actions

pre_read_actions_redefinition:
    = 'DELETE' 'PRE_READ_ACTIONS' ';'
    = 'REDEFINE' pre_read_actions

pre_write_actions_redefinition:
    = 'DELETE' 'PRE_WRITE_ACTIONS' ';'
    = 'REDEFINE' pre_write_actions

read_timeout_redefinition:
    = 'DELETE' 'READ_TIMEOUT' ';'
    = 'REDEFINE' read_timeout

refresh_actions_redefinition:
    = 'DELETE' 'REFRESH_ACTIONS' ';'
    = 'REDEFINE' refresh_actions

variable_style_redefinition:
    = 'DELETE' 'STYLE' ';'
    = 'REDEFINE' variable_style

write_timeout_redefinition:
    = 'DELETE' 'WRITE_TIMEOUT' ';'
    = 'REDEFINE' write_timeout

variable_list_redefinition:
    = 'DELETE' 'VARIABLE_LIST' Identifier ';'

```

```

= 'REDEFINE' 'VARIABLE_LIST' Identifier '{' '}'
= 'REDEFINE' 'VARIABLE_LIST' Identifier '{'
  variable_list_attribute_list '}'
= 'VARIABLE_LIST' Identifier '{'
  variable_list_attribute_redefinition_list '}'

variable_list_attribute_redefinition_list:
= variable_list_attribute_redefinition
= variable_list_attribute_redefinition_list
  variable_list_attribute_redefinition

variable_list_attribute_redefinition:
= members_redefinition
= help_redefinition
= optional_label_redefinition
= response_codes_reference_redefinition

waveform_redefinition:
= 'DELETE' 'WAVEFORM' Identifier ';'
= 'REDEFINE' 'WAVEFORM' Identifier '{' waveform_attribute_list '}'
= 'WAVEFORM' Identifier '{' waveform_attribute_redefinition_list '}'

waveform_attribute_redefinition_list:
= waveform_attribute_redefinition
= waveform_attribute_redefinition_list waveform_attribute_redefinition

waveform_attribute_redefinition:
= waveform_type_redefinition
= emphasis_redefinition
= waveform_exit_actions_redefinition
= handling_redefinition
= help_redefinition
= waveform_init_actions_redefinition
= waveform_key_points_redefinition
= optional_label_redefinition
= line_color_redefinition
= line_type_redefinition
= waveform_refresh_actions_redefinition
= validity_redefinition
= waveform_y_axis_redefinition

waveform_type_redefinition:
= 'REDEFINE' waveform_type
= 'TYPE' waveform_type_option_redefinition

waveform_type_option_redefinition:
= 'YT' '{' waveform_yt_type_option_redefinition_list '}'
= 'XY' '{' waveform_xy_type_option_redefinition_list '}'
= 'HORIZONTAL' '{' waveform_y_values_redefinition '}'
= 'VERTICAL' '{' waveform_x_values_redefinition '}'

waveform_yt_type_option_redefinition_list:
= waveform_yt_type_option_redefinition
= waveform_yt_type_option_redefinition_list
  waveform_yt_type_option_redefinition

waveform_yt_type_option_redefinition:
= yt_x_initial_redefinition
= yt_x_increment_redefinition
= waveform_y_values_redefinition
= number_of_points_redefinition

yt_x_initial_redefinition:

```

```

    = 'REDEFINE' yt_x_initial

yt_x_increment_redefinition:
    = 'REDEFINE' yt_x_increment

waveform_y_values_redefinition:
    = 'REDEFINE' waveform_y_values
    = 'ADD' waveform_y_values

waveform_x_values_redefinition:
    = 'REDEFINE' waveform_x_values
    = 'ADD' waveform_x_values

number_of_points_redefinition:
    = 'DELETE' 'NUMBER_OF_POINTS' ';'
    = 'REDEFINE' number_of_points

waveform_xy_type_option_redefinition_list:
    = waveform_xy_type_option_redefinition
    = waveform_xy_type_option_redefinition_list
    waveform_xy_type_option_redefinition

waveform_xy_type_option_redefinition:
    = waveform_x_values_redefinition
    = waveform_y_values_redefinition
    = number_of_points_redefinition

waveform_exit_actions_redefinition:
    = 'DELETE' 'EXIT_ACTIONS' ';'
    = 'REDEFINE' waveform_exit_actions

waveform_init_actions_redefinition:
    = 'DELETE' 'INIT_ACTIONS' ';'
    = 'REDEFINE' waveform_init_actions

waveform_key_points_redefinition:
    = 'DELETE' 'KEY_POINTS' ';'
    = 'REDEFINE' 'KEY_POINTS' '{' key_points_options_redefinition_list '}'

key_points_options_redefinition_list:
    = key_points_options_redefinition
    = key_points_options_redefinition_list key_points_options_redefinition

key_points_options_redefinition:
    = waveform_x_values_redefinition
    = waveform_y_values_redefinition

waveform_refresh_actions_redefinition:
    = 'DELETE' 'REFRESH_ACTIONS' ';'
    = 'REDEFINE' waveform_refresh_actions

waveform_y_axis_redefinition:
    = 'DELETE' 'Y_AXIS' ';'
    = 'REDEFINE' waveform_y_axis

```

A.6.40 References

```

reference:
    = reference_without_call
    = reference_without_call '(' argument_list ')'

reference_without_call:
    = Identifier
    = reference_without_call '[' expr ']'

```

```

= reference_without_call '.' Identifier
= reference_without_call '.' selector
= reference_without_call '->' Identifier
= 'BLOCK' '.' Identifier
= 'LOCAL_PARAM' '.' Identifier
= 'PARAM' '.' Identifier
= 'PARAM_LIST' '.' Identifier
= block_a_reference '[' expr ']' '.' 'PARAM' '.' Identifier
= block_a_reference '[' expr ']' '.' 'LOCAL_PARAM' '.' Identifier
= block_a_reference '[' expr ']' '.' 'BLOCK' '.' Identifier
= block_a_reference '[' expr ']' '.' 'CHART' '.' Identifier
= block_a_reference '[' expr ']' '.' 'LIST' '.' Identifier
= block_a_reference '[' expr ']' '.' 'GRAPH' '.' Identifier
= block_a_reference '[' expr ']' '.' 'GRID' '.' Identifier
= block_a_reference '[' expr ']' '.' 'MENU' '.' Identifier
= block_a_reference '[' expr ']' '.' 'METHOD' '.' Identifier

```

selector:

```

= 'CAPACITY'
= 'CONSTANT_UNIT'
= 'COUNT'
= 'DEFAULT_VALUE'
= 'FIRST'
= 'HELP'
= 'INITIAL_VALUE'
= 'LABEL'
= 'LAST'
= 'MAX_VALUE'
= 'MAX_VALUE1'
= 'MAX_VALUE2'
= 'MAX_VALUE3'
= 'MAX_VALUE4'
= 'MAX_VALUE5'
= 'MAX_VALUE6'
= 'MAX_VALUE7'
= 'MAX_VALUE8'
= 'MAX_VALUE9'
= 'MIN_VALUE'
= 'MIN_VALUE1'
= 'MIN_VALUE2'
= 'MIN_VALUE3'
= 'MIN_VALUE4'
= 'MIN_VALUE5'
= 'MIN_VALUE6'
= 'MIN_VALUE7'
= 'MIN_VALUE8'
= 'MIN_VALUE9'
= 'SCALING'
= 'SCALING_FACTOR'
= 'VARIABLE_STATUS'
= 'VIEW_MAX'
= 'VIEW_MIN'
= 'X_AXIS'
= 'Y_AXIS'

```

dictionary_reference:

```

= '[' Identifier ']'

```

array_reference:

```

= reference_without_call

```

axis_reference:

```

= reference_without_call

```

```

block_a_reference:
    = reference_without_call

block_b_reference:
    = reference_without_call

chart_reference:
    = reference_without_call

collection_reference:
    = reference_without_call

connection_reference:
    = reference_without_call

edit_display_reference:
    = reference_without_call

file_reference:
    = reference_without_call

graph_reference:
    = reference_without_call

grid_reference:
    = reference_without_call

image_reference:
    = reference_without_call

list_reference:
    = reference_without_call

item_array_reference:
    = reference_without_call

menu_reference:
    = reference_without_call

method_reference:
    = reference

record_reference:
    = reference_without_call

refresh_reference:
    = reference_without_call

response_code_reference:
    = reference_without_call

source_reference:
    = reference_without_call

unit_reference:
    = reference_without_call

variable_reference:
    = reference_without_call

wao_reference:
    = reference_without_call

```



```
waveform_reference:  
    = reference_without_call
```

IECNORM.COM: Click to view the full PDF of IEC 61804-3:2010

Withd2Wn

Annex B (normative)

EDDL Builtin library

B.1 General

This annex describes Builtin subroutines, which can be used in an EDD. The Builtins support user or device interactions. Typical user interactions are, for example, user inputs of VARIABLE values or acknowledgments. Typical device interactions are, for example, send commands to the device or interpreting response codes and status bytes. This annex contains the library of Builtins.

In addition to the Builtins specified in this document, user-specific Builtins can be used. These Builtins shall be integrated into the EDD application.

NOTE The Builtins are listed in alphabetical order.

B.2 Conventions for Builtin descriptions

The following conventions apply for the Builtin definitions.

- Table B.1 defines the format for the Builtins lexical element tables.
- Table B.2 defines the contents of the columns of Table B.1.

Table B.1 – Format for the Builtins lexical element tables

Parameter Name	Type	Direction	Usage	Description

Table B.2 – Contents of the lexical element table

Column	Text	Meaning
Parameter name	<name>	The name of the formal parameter.
	"<return>"	The return value of the Builtin.
Type	data type	Abstract data type (basic or derived) of the formal parameter.
Direction	I	An input parameter.
	O	An output parameter.
	I/O	An input and output parameter.
	—	There is no returned value (VOID).
Usage	m	Formal parameter is mandatory.
	o	Formal parameter is optional.
	c	Formal parameter is conditional.
	—	There is no returned value (VOID).
Description	<text>	Descriptive text for the parameter.
	—	There is no returned value (VOID).

B.3 Builtin _ERROR

Purpose

The Builtin _ERROR puts a textual information in a log. This information is put into the category error. The Builtin _TRACE and the Builtin _WARNING write information to this same log.

Any variable reference contained in the format string will be expanded to the current value. Method local variables and DD variables can be referenced in the message parameter. For more information, refer to the Builtin PUT_MESSAGE.

Lexical structure

`_WARNING(text, global_var_ids)`

The lexical elements of the Builtin _TRACE are shown in Table B.3.

Table B.3 – Builtin _ERROR

Parameter name	Type	Direction	Usage	Description
text	DD_STRING	I	m	The textual information to be put into the log.
global_var_ids	int[]	I	m	An array which contains unique identifiers of VARIABLE instance.
<return>	VOID	—	—	—

B.4 Builtin _TRACE

Purpose

The Builtin _TRACE puts a textual information in a log. This information is put into the category trace. The Builtin _ERROR and the Builtin _WARNING write information to this same log.

Any variable reference contained in the format string will be expanded to the current value. Method local variables and DD variables can be referenced in the message parameter. For more information, refer to the Builtin PUT_MESSAGE.

Lexical structure

`_TRACE(text, global_var_ids)`

The lexical elements of the Builtin _TRACE are shown in Table B.4.

Table B.4 – Builtin _TRACE

Parameter name	Type	Direction	Usage	Description
text	DD_STRING	I	m	The textual information to be put into the log.
global_var_ids	int[]	I	m	An array which contains unique identifiers of VARIABLE instance.
<return>	VOID	—	—	—

B.5 Builtin _WARNING

Purpose

The Builtin _WARNING puts a textual information in a log. This information is put into the category warning. The Builtin _ERROR and the Builtin _TRACE write information to this same log.

Any variable reference contained in the format string will be expanded to the current value. Method local variables and DD variables can be referenced in the message parameter. For more information, refer to the Builtin PUT_MESSAGE.

Lexical structure

`_WARNING(text, global_var_ids)`

The lexical elements of the Builtin _TRACE are shown in Table B.5.

Table B.5 – Builtin _WARNING

Parameter name	Type	Direction	Usage	Description
text	DD_STRING	I	m	The textual information to be put into the log.
global_var_ids	int[]	I	m	An array which contains unique identifiers of VARIABLE instance.
<return>	VOID	—	—	—

B.6 Builtin abort

Purpose

The Builtin abort will display a message indicating that the method has been aborted and wait for acknowledgement from the user. Once acknowledgement has been made, the system will execute any abort methods in the abort method list and will exit the method.

Lexical structure

abort (VOID)

The lexical elements of the Builtin abort are shown in Table B.6.

Table B.6 – Builtin abort

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.7 Builtin abort_on_all_comm_errors

Purpose

The Builtin abort_on_all_comm_errors sets the method to abort upon receiving any communication error.

When a method is started, it does not abort on a communication error until abort_on_all_comm_errors is called. This frees a method from having to handle any communication error codes by aborting the method if it receives any communications error.

After this call, any Builtin call which makes a communication request that returns an error will cause the method to abort. A communication error of zero is not considered a communication error.

The communication error codes are defined by the consortia. Their related strings may be defined in a text dictionary.

Lexical structure

abort_on_all_comm_errors (VOID)

The lexical element of the Builtin abort_on_all_comm_errors is shown in Table B.7.

Table B.7 – Builtin abort_on_all_comm_errors

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.8 Builtin ABORT_ON_ALL_COMM_STATUS

Purpose

The Builtin ABORT_ON_ALL_COMM_STATUS will set all of the bits in the comm status abort mask. This will cause the system to abort the current method if the device returns any comm

status value. Comm status is defined to be the first data octet returned in a transaction, when bit 7 of this octet is set.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See Builtin ABORT_ON_RESPONSE_CODES for a list of the available masks, and their default values.

The mask affected by this function is used by the Builtins send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

ABORT_ON_ALL_COMM_STATUS (VOID)

The lexical element of the Builtin ABORT_ON_ALL_COMM_STATUS is shown in Table B.8.

Table B.8 – Builtin ABORT_ON_ALL_COMM_STATUS

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.9 Builtin ABORT_ON_ALL_DEVICE_STATUS

Purpose

The Builtin ABORT_ON_ALL_DEVICE_STATUS will set all of the bits in the device status abort mask. This will cause the system to abort the current method if the device returns any device status value. Device status is defined to be the second data octet returned in a transaction.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this function is used by the Builtins send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

ABORT_ON_ALL_DEVICE_STATUS (VOID)

The lexical element of the Builtin ABORT_ON_ALL_DEVICE_STATUS is shown in Table B.9.

Table B.9 – Builtin ABORT_ON_ALL_DEVICE_STATUS

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.10 Builtin ABORT_ON_ALL_RESPONSE_CODES

Purpose

The Builtin ABORT_ON_ALL_RESPONSE_CODES will set all of the bits in the response code abort mask. This will cause the system to abort the current method if the device returns any response code value.

The response code is defined to be the first data octet returned in a transaction, when bit 7 of this octet is 0.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send command function for implementation of the masks. See ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this function is used by the Builtins send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

ABORT_ON_ALL_RESPONSE_CODES (VOID)

The lexical element of the Builtin ABORT_ON_ALL_RESPONSE_CODES is shown in Table B.10.

Table B.10 – Builtin ABORT_ON_ALL_RESPONSE_CODES

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.11 Builtin abort_on_all_response_codes

Purpose

The Builtin abort_on_all_response_codes causes the method to abort if it receives any non-zero response code, after calling a Builtin, which results in a request to a device. This Builtin frees a method from having to handle any error response codes. When a method is started, it does not abort on any response code until this Builtin is called. A response code is a value representing an application-specific error condition. A response code value of zero is not considered as an error response code. Each response code and response code type is defined in the EDDL.

Lexical structure

abort_on_all_response_codes (VOID)

The lexical element of the Builtin abort_on_all_response_codes is shown in Table B.11.

Table B.11 – Builtin abort_on_all_response_codes

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.12 Builtin ABORT_ON_COMM_ERROR

Purpose

The Builtin ABORT_ON_COMM_ERROR will set the no comm error mask in such a way that the method will be aborted if a comm error is found while sending a command.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODES for a list of the available masks and their default values.

The mask affected by this function is used by the Builtins send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

ABORT_ON_COMM_ERROR (VOID)

The lexical element of the Builtin ABORT_ON_COMM_ERROR is shown in Table B.12.

Table B.12 – Builtin ABORT_ON_COMM_ERROR

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.13 Builtin abort_on_comm_error

Purpose

The Builtin abort_on_comm_error causes the method to abort if it receives a specific communications error.

When a method is started, it does not abort on a specific communications error until Builtin abort_on_comm_error is called.

NOTE A communications error of zero is not considered a communications error.

This Builtin permits the method writer to select which communications errors should be handled by the method and it frees a method from having to handle a specific communications error code.

The communications error codes are defined by the consortia. Their related strings may be defined in a text dictionary.

Lexical structure

abort_on_comm_error(error)

The lexical elements of the Builtin abort_on_comm_error are shown in Table B.13.

Table B.13 – Builtin abort_on_comm_error

Parameter name	Type	Direction	Usage	Description
error	unsigned long	I	m	A communications error code.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.14 Builtin ABORT_ON_COMM_STATUS

Purpose

The Builtin ABORT_ON_COMM_STATUS will set the correct bit(s) in the comm status abort mask in such a way that the specified comm_status_value will cause the method to abort. Comm status is defined to be the first data octet returned in a transaction, when bit 7 of this octet is 1.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send command function for implementation of the masks. See ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this function is used by the Builtins send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

ABORT_ON_COMM_STATUS(comm_status)

The lexical elements of the Builtin ABORT_ON_COMM_STATUS are shown in Table B.14.

Table B.14 – Builtin ABORT_ON_COMM_STATUS

Parameter name	Type	Direction	Usage	Description
comm_status	unsigned char	l	m	A bit mask for the communication status.
<return>	VOID	—	—	—

B.15 Builtin ABORT_ON_DEVICE_STATUS

Purpose

The Builtin ABORT_ON_DEVICE_STATUS will set the correct bit(s) in the device status abort mask such that the specified device status value will cause the method to abort. Device status is defined to be the second data octet returned in a transaction.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send command function for implementation of the masks. See ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this function is used by the Builtins send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

ABORT_ON_DEVICE_STATUS(device_status)

The lexical elements of the Builtin ABORT_ON_DEVICE_STATUS are shown in Table B.15.

Table B.15 – Builtin ABORT_ON_DEVICE_STATUS

Parameter name	Type	Direction	Usage	Description
device_status	unsigned char	l	m	A bit mask for the device status.
<return>	VOID	—	—	—

B.16 Builtin ABORT_ON_NO_DEVICE

Purpose

The Builtin ABORT_ON_NO_DEVICE will set the no devices mask in such a way that the method will be aborted if no device is found while sending a command.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send command function for implementation of the masks. See ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this function is used by the Builtins send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

ABORT_ON_NO_DEVICE(VOID)

The lexical element of the Builtin ABORT_ON_NO_DEVICE is shown in Table B.16.

Table B.16 – Builtin ABORT_ON_NO_DEVICE

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.17 Builtin ABORT_ON_RESPONSE_CODE

Purpose

The Builtin ABORT_ON_RESPONSE_CODE will set the correct bit(s) in the response code abort mask such that the specified response code value will cause the method to abort. The response code is defined to be the first data octet returned in a transaction, when bit 7 of this octet is 0.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method.

The following is a list of all the available masks. The default values of the masks are as follows:

- response_code_abort_mask = ACCESS_RESTRICTED | CMD_NOT_IMPLEMENTED
- response_code_retry_mask = NO BITS SET
- cmd_48_response_code_abort_mask = ACCESS_RESTRICTED | CMD_NOT_IMPLEMENTED
- cmd_48_response_code_retry_mask = NO BITS SET
- device_status_abort_mask = DEVICE_MALFUNCTION
- device_status_retry_mask = NO BITS SET
- cmd_48_device_status_abort_mask = DEVICE_MALFUNCTION
- cmd_48_device_status_retry_mask = NO BITS SET
- comm_status_abort_mask = NO BITS SET
- comm_status_retry_mask = ALL BITS SET
- cmd_48_comm_status_abort_mask = NO BITS SET
- cmd_48_comm_status_retry_mask = ALL BITS SET
- no_device_abort_mask = ALL BITS SET
- comm_error_abort_mask = ALL BITS SET
- comm_error_retry_mask = NO BITS SET
- cmd_48_no_device_abort_mask = ALL BITS SET
- cmd_48_comm_error_abort_mask = ALL BITS SET
- cmd_48_comm_error_retry_mask = NO BITS SET
- cmd_48_data_abort_mask = NO BITS SET
- cmd_48_data_retry_mask = NO BITS SET

The mask affected by this function is used by the Builtins send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

ABORT_ON_RESPONSE_CODE(response_code)

The lexical elements of the Builtin ABORT_ON_RESPONSE_CODE are shown in Table B.17.

Table B.17 – Builtin ABORT_ON_RESPONSE_CODE

Parameter name	Type	Direction	Usage	Description
response_code	int	I	m	A bit mask for the response code.
<return>	VOID	—	—	—

B.18 Builtin abort_on_response_code**Purpose**

The Builtin abort_on_response_code causes the method to abort upon receiving the specified response code.

The Builtin causes the method to abort if it receives a specific response code. This Builtin frees a method from having to handle a specific response code.

When a method is started, it will not abort on a specific response code until this Builtin is called. A response code is a value representing an application-specific error condition.

NOTE A response code value of zero is not considered an error response code. Each response code type is defined in the EDD.

Lexical structure

abort_on_response_code (code)

The lexical elements of the Builtin abort_on_response_code are shown in Table B.18.

Table B.18 – Builtin abort_on_response_code

Parameter name	Type	Direction	Usage	Description
code	unsigned long	I	m	A bit mask for the response code.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.19 Builtin abs**Purpose**

The Builtin abs returns the absolute value of a floating point value.

Lexical structure

abs (x)

The lexical elements of the Builtin abs are shown in Table B.19.

Table B.19 – Builtin abs

Parameter name	Type	Direction	Usage	Description
x	double	I	m	A floating-point value.
<return>	double	O	m	The absolute value.

B.20 Builtin ACKNOWLEDGE**Purpose**

The Builtin ACKNOWLEDGE will display the prompt and wait for the enter key to be pressed. Unlike Builtin acknowledge, the prompt may NOT contain embedded device VARIABLES.

Lexical structure

ACKNOWLEDGE (prompt)

The lexical elements of the Builtin ACKNOWLEDGE are shown in Table B.20.

Table B.20 – Builtin ACKNOWLEDGE

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	Message to be displayed.
<return>	int	O	m	One of the return codes specified in Table B.347.

B.21 Builtin acknowledge

Purpose

The Builtin acknowledge will display the prompt and wait for acknowledgment from the user. The prompt may contain local and/or device variable values.

Lexical structure

`acknowledge(prompt, global_var_ids)`

The lexical elements of the Builtin acknowledge are shown in Table B.21.

Table B.21 – Builtin acknowledge

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	Message to be displayed.
global_var_ids	array of int	I	m	An array that contains unique identifiers of VARIABLE instances.
<return>	int	O	m	One of the return codes specified in Table B.347.

B.22 Builtin acos

Purpose

The Builtin acos returns the arc cosine of a floating-point value.

Lexical structure

`acos(x)`

The lexical elements of the Builtin acos are shown in Table B.22.

Table B.22 – Builtin acos

Parameter name	Type	Direction	Usage	Description
x	double	I	m	A floating-point value.
<return>	double	O	m	The arc cosine.

B.23 Builtin add_abort_method (version A)

Purpose

The Builtin add_abort_method will add a method to the abort method list, which is the list of methods to be executed if the current method is aborted. The abort method list can hold up to 20 methods at any one time. The methods are run in the order they are added to the list, and the same method may be added to the list more than once. The list is cleared after each non_abort method is executed.

add_abort_method is only executed when the method is aborted, and not when the method exits under normal operating conditions. Methods can be aborted due to an abort mask condition when sending a command, or when the Builtin abort is called.

Lexical structure

`add_abort_method(abort_method_id)`

The lexical elements of the Builtin `add_abort_method` are shown in Table B.23.

Table B.23 – Builtin `add_abort_method`

Parameter name	Type	Direction	Usage	Description
<code>abort_method_id</code>	reference	I	m	The name of a METHOD instance.
<return>	int	O	m	One of the return codes specified in Table B.347.

B.24 Builtin `add_abort_method` (version B)

Purpose

The Builtin `add_abort_method` adds an abort method to the calling method's abort method list. The abort method list is the list of methods, which will be executed if the method aborts. The abort method list applies only to the current execution of the method.

Methods abort when response codes or communication errors occur which have been set to trigger an abort via other Builtin calls, for example, `abort_on_all_comm_error` and `abort_on_all_response_codes`. Methods can also be explicitly aborted by a direct call to Builtin method `_abort`, or when an error occurs. The normal completion of a method is not considered an abort.

The abort method list is initially empty when a method begins execution. The abort methods are added to the back of the list and are removed and run from the front of the list in the order they appear, i.e. first-in first-out. An abort method may appear more than once in the list.

Abort methods may be shared between methods, meaning two methods may have the same abort method in their respective abort method lists.

The `method_id` parameter is specified by the method writer using the Builtin `ITEM_ID` and the name of the method. Builtin abort methods cannot call `add_abort_method`.

Lexical structure

`add_abort_method(method_id)`

The lexical elements of the Builtin `add_abort_method` are shown in Table B.24.

Table B.24 – Builtin `add_abort_method`

Parameter name	Type	Direction	Usage	Description
<code>method_id</code>	unsigned long	I	m	The ID of the method.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.25 Builtin `AddTime`

Purpose

The Builtin `AddTime` adds a number of seconds to a `time_t` value.

Lexical Structure

`AddTime(t0, seconds)`

The lexical elements of the Builtin `AddTime` are shown in Table B.25.

Table B.25 – Builtin AddTime

Parameter name	Type	Direction	Usage	Description
t0	time_t	I	m	The initial time_t value.
seconds	double	I	m	The number of seconds to add.
<return>	time_t	O	m	The specified time_t value.

B.26 Builtin asin**Purpose**

The Builtin asin returns the arc sine of a floating point value.

Lexical structure

asin(x)

The lexical elements of the Builtin asin are shown in Table B.26.

Table B.26 – Builtin asin

Parameter name	Type	Direction	Usage	Description
x	double	I	m	A floating point value.
<return>	double	O	m	The arc sine.

B.27 Builtin assign**Purpose**

The Builtin assign sets the value of a VARIABLE to another VARIABLE's value. It sets the value of the destination VARIABLE equal to the value of the source VARIABLE.

NOTE The Builtin is a shorthand way of doing an ABN_GET_TYPE_VALUE followed by a put_type_value.

The parameters dst_id, src_id, dst_member_id, and src_member_id are specified using the Builtins ITEM_ID and MEMBER_ID and the reference of the VARIABLE. Both the source VARIABLES and the destination VARIABLE shall have the same type.

Lexical structure

assign(dst_id, dst_member_id, src_id, src_member_id)

The lexical elements of the Builtin assign are shown in Table B.27.

Table B.27 – Builtin assign

Parameter name	Type	Direction	Usage	Description
dst_id	unsigned long	I	m	The item ID of the destination variable.
dst_member_id	unsigned long	I	m	The member ID of the destination variable.
src_id	unsigned long	I	m	The item ID of the source variable.
src_member_id	unsigned long	I	m	The member ID of the source variable.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.28 Builtin assign_double

Purpose

The Builtin assign_double assigns a value to a VARIABLE.

Lexical structure

assign_double(device_var, new_value)

The lexical elements of the Builtin assign_double are shown in Table B.28.

Table B.28 – Builtin assign_double

Parameter name	Type	Direction	Usage	Description
device_var	reference	I	m	The name of the variable.
new_value	double	I	m	The new value of the referenced variable.
<return>	int	O	m	One of the return codes specified in Table B.347.

B.29 Builtin assign_float

Purpose

The Builtin assign_float assigns a value to VARIABLE.

Lexical structure

assign_float(device_var, new_value)

The lexical elements of the Builtin assign_float are shown in Table B.29.

Table B.29 – Builtin assign_float

Parameter name	Type	Direction	Usage	Description
device_var	reference	I	m	The name of the variable.
new_value	double	I	m	The new value of the referenced variable.
<return>	int	O	m	One of the return codes specified in Table B.347.

B.30 Builtin assign_int

Purpose

The Builtin assign_int assigns a value to VARIABLE.

Lexical structure

assign_int(device_var, new_value)

The lexical elements of the Builtin assign_int are shown in Table B.30.

Table B.30 – Builtin assign_int

Parameter name	Type	Direction	Usage	Description
device_var	reference	I	m	The name of the variable.
new_value	int	I	m	The new value of the referenced variable.
<return>	int	O	m	One of the return codes specified in Table B.347.

B.31 Builtin assign_var

Purpose

The Builtin assign_var assigns a value of the source VARIABLE to the destination VARIABLE. Both variables shall have the same type.

Lexical structure

```
assign_var(destination_var, source_var)
```

The lexical elements of the Builtin assign_var are shown in Table B.31.

Table B.31 – Builtin assign_var

Parameter name	Type	Direction	Usage	Description
destination_var	reference	I	m	The name of the destination variable.
source_var	reference	I	m	The name of the source variable.
<return>	int	O	m	One of the return codes specified in Table B.347.

B.32 Builtin assign2

Purpose

The Builtin assign2 sets the value of a VARIABLE to another VARIABLE's value. It sets the value of the destination VARIABLE equal to the value of the source VARIABLE.

The parameters dst_id, src_id, dst_member_id, and src_member_id are specified using the Builtins ITEM_ID and MEMBER ID and the reference of the VARIABLE. Both the source VARIABLES and the destination VARIABLE shall have the same type.

Lexical structure

```
assign2(dst_blk_id, dst_blk_num, dst_id, dst_member_id, src_blk_id, src_blk_num, src_id, src_member_id)
```

The lexical elements of the Builtin assign2 are shown in Table B.32.

Table B.32 – Builtin assign2

Parameter name	Type	Direction	Usage	Description
dst_blk_id	unsigned long	I	m	The item ID of the destination block.
dst_blk_num	unsigned long	I	m	The occurrence number of the destination block.
dst_id	unsigned long	I	m	The item ID of the destination variable.
dst_member_id	unsigned long	I	m	The member ID of the destination variable.
src_blk_id	unsigned long	I	m	The item ID of the source block.
src_blk_num	unsigned long	I	m	The occurrence number of the source block.
src_id	unsigned long	I	m	The item ID of the source variable.
src_member_id	unsigned long	I	m	The member ID of the source variable.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.33 Builtin atan

Purpose

The Builtin atan returns the arc tangent of a floating point value.

Lexical structure

`atan(x)`

The lexical elements of the Builtin atan are shown in Table B.33.

Table B.33 – Builtin atan

Parameter Name	Type	Direction	Usage	Description
x	double	I	m	A floating point value.
<return>	double	O	m	The arc tangent.

B.34 Builtin ATOF

Purpose

The Builtin ATOF converts a string into a floating-point value.

Lexical structure

`ATOF(string)`

The lexical elements of the Builtin ATOF are shown in Table B.34.

Table B.34 – Builtin ATOF

Parameter name	Type	Direction	Usage	Description
string	DD_STRING	I	m	The string to convert.
<return>	double	O	m	The floating point value.

B.35 Builtin atof

Purpose

The Builtin converts a text into a floating-point value.

Lexical structure

`atof(str)`

The lexical elements of the Builtin atof are shown in Table B.35.

Table B.35 – Builtin atof

Parameter name	Type	Direction	Usage	Description
str	char[]	I	m	The text to convert.
<return>	float	O	m	The floating point value.

B.36 Builtin ATOI

Purpose

The Builtin ATOI converts a string into an integer value.

Lexical structure

`ATOI(str)`

The lexical elements of the Builtin ATOI are shown in Table B.36.

Table B.36 – Builtin ATOI

Parameter name	Type	Direction	Usage	Description
str	DD_STRING	I	m	The string to convert.
<return>	int	O	m	The integer value.

B.37 Builtin atoi

Purpose

The Builtin converts a text into an integer value.

Lexical structure

`atoi(str)`

The lexical elements of the Builtin atoi are shown in Table B.37.

Table B.37 – Builtin atoi

Parameter name	Type	Direction	Usage	Description
str	char[]	I	m	The text to convert.
<return>	int	O	m	The integer value.

B.38 Builtin BUILD_MESSAGE

Purpose

The Builtin BUILD_MESSAGE returns a string based on a format specification. Any variable reference contained in the format string will be expanded to the current value. Method local variables and DD variables can be referenced in the message parameter. For more information, refer to the Builtin PUT_MESSAGE.

Lexical structure

`BUILD_MESSAGE(string)`

The lexical elements of the Builtin BUILD_MESSAGE are shown in Table B.38.

Table B.38 – Builtin BUILD_MESSAGE

Parameter name	Type	Direction	Usage	Description
string	DD_STRING	I	m	The format specification.
<return>	DD_STRING	O	m	The formatted string.

B.39 Builtin ByteToDouble

Purpose

The Builtin ByteToDouble will return a double value from a vector of eight octets.

Lexical structure

`ByteToDouble(in1, in2, in3, in4, in5, in6, in7, in8)`

The lexical element of the Builtin ByteToDouble is shown in Table B.39.

Table B.39 – Builtin ByteToDouble

Parameter name	Type	Direction	Usage	Description
in1	unsigned char	I	m	First element of the double value (most significant byte).
in2	unsigned char	I	m	Second element of the double value.
in3	unsigned char	I	m	Third element of the double value.
in4	unsigned char	I	m	Fourth element of the double value.
in5	unsigned char	I	m	Fifth element of the double value.
in6	unsigned char	I	m	Sixth element of the double value.
in7	unsigned char	I	m	Seventh element of the double value.
in8	unsigned char	I	m	Eighth element of the double value (least significant byte).
<return>	double	O	m	One of the return codes specified in Table B.347.

B.40 Builtin ByteToFloat**Purpose**

The Builtin ByteToFloat will return a floating value from a vector of four octets.

Lexical structure

ByteToFloat(in1, in2, in3, in4)

The lexical element of the Builtin ByteToFloat is shown in Table B.40.

Table B.40 – Builtin ByteToFloat

Parameter name	Type	Direction	Usage	Description
in1	unsigned char	I	m	First element of the float value (most significant byte).
in2	unsigned char	I	m	Second element of the float value.
in3	unsigned char	I	m	Third element of the float value.
in4	unsigned char	I	m	Fourth element of the float value (least significant byte).
<return>	float	O	m	One of the return codes specified in Table B.347.

B.41 Builtin ByteToLong**Purpose**

The Builtin ByteToLong will return a long value from a vector of four octets.

Lexical structure

ByteToLong(in1, in2, in3, in4)

The lexical element of the Builtin ByteToLong is shown in Table B.41.

Table B.41 – Builtin ByteToLong

Parameter name	Type	Direction	Usage	Description
in1	unsigned char	I	m	First element of the long value (most significant byte).
in2	unsigned char	I	m	Second element of the long value.
in3	unsigned char	I	m	Third element of the long value.
in4	unsigned char	I	m	Fourth element of the long value (least significant byte).
<return>	long	O	m	One of the return codes specified in Table B.347.

B.42 Builtin ByteToShort**Purpose**

The Builtin ByteToShort will return a short value from a vector of two octets.

Lexical structure

```
ByteToShort(in1, in2)
```

The lexical element of the Builtin ByteToShort is shown in Table B.42.

Table B.42 – Builtin ByteToShort

Parameter name	Type	Direction	Usage	Description
in1	unsigned char	I	m	First element of the short value (most significant byte).
in2	unsigned char	I	m	Second element of the short value (least significant byte).
<return>	short	O	m	One of the return codes specified in Table B.347.

B.43 Builtin cbrt**Purpose**

The Builtin cbrt returns the cubic root of a floating-point value.

Lexical structure

```
cbrt(x)
```

The lexical elements of the Builtin cbrt are shown in Table B.43.

Table B.43 – Builtin cbrt

Parameter name	Type	Direction	Usage	Description
x	double	I	m	A floating-point value.
<return>	double	O	m	The cubic root.

B.44 Builtin ceil**Purpose**

The Builtin ceil returns the smallest integral value not less than a specified floating-point value.

Lexical structure

`ceil(x)`

The lexical elements of the Builtin `ceil` are shown in Table B.44.

Table B.44 – Builtin `ceil`

Parameter name	Type	Direction	Usage	Description
<code>x</code>	double	I	m	A floating-point value.
<return>	double	O	m	The ceiling value.

B.45 Builtin `cos`

Purpose

The Builtin `cos` returns the cosine of a floating point value.

Lexical structure

`cos(x)`

The lexical elements of the Builtin `cos` are shown in Table B.45.

Table B.45 – Builtin `cos`

Parameter name	Type	Direction	Usage	Description
<code>x</code>	double	I	m	A floating point value.
<return>	double	O	m	The cosine.

B.46 Builtin `cosh`

Purpose

The Builtin `cosh` returns the hyperbolic cosine of a floating point value.

Lexical structure

`cosh(x)`

The lexical elements of the Builtin `cosh` are shown in Table B.46.

Table B.46 – Builtin `cosh`

Parameter name	Type	Direction	Usage	Description
<code>x</code>	double	I	m	A floating-point value.
<return>	double	O	m	The hyperbolic cosine.

B.47 Builtin `dassign`

Purpose

The Builtin `dassign` assigns a value to a VARIABLE.

Lexical structure

`dassign(device_var, new_value)`

The lexical elements of the Builtin `dassign` are shown in Table B.47.

Table B.47 – Builtin dassign

Parameter name	Type	Direction	Usage	Description
device_var	reference	I	m	The name of the variable.
new_value	double	I	m	The new value of the referenced variable.
<return>	int	O	m	One of the return codes specified in Table B.347.

B.48 Builtin DATE_AND_TIME_VALUE_to_string**Purpose**

The Builtin DATE_AND_TIME_VALUE_to_string creates a string representation of a DATE and a TIME_VALUE. The ANSI C strftime function defines the structure of the format string.

Lexical structure

DATE_AND_TIME_VALUE_to_string(string, format, date, time_value)

The lexical element of the Builtin DATE_AND_TIME_VALUE_to_string is shown in Table B.48.

Table B.48 – Builtin DATE_AND_TIME_VALUE_to_string

Parameter name	Type	Direction	Usage	Description
string	DD_STRING	I	m	The string into which formatted DATE and TIME_VALUE are copied.
format	DD_STRING	I	m	How the DATE and TIME_VALUE are to be formatted.
date	long	I	m	The DATE to be formatted.
time_value	unsigned long	I	M	The TIME_VALUE to be formatted.
<return>	int	O	m	The number of character actually copied or -1 is an error occurs.

B.49 Builtin Date_to_DayOfMonth**Purpose**

The Builtin Date_to_DayOfMonth calculates the day of the month from a given number of days since 1 January 1900.

Lexical structure

Date_to_DayOfMonth(date)

The lexical elements of the Builtin Date_to_DayOfMonth are shown in Table B.49.

Table B.49 – Builtin Date_to_DayOfMonth

Parameter name	Type	Direction	Usage	Description
date	long	I	m	The date. The EDD application shall support an implicit typecast between long and DATE, so this Builtin will also accept DATE types.
<return>	int	O	m	The day of month.

B.50 Builtin DATE_to_days**Purpose**

The Builtin DATE_to_days calculates the number of days between two dates.

Lexical structure

DATE_to_days(date1, date0)

The lexical element of the Builtin DATE_to_days is shown in Table B.50.

Table B.50 – Builtin DATE_to_days

Parameter name	Type	Direction	Usage	Description
date1	long	I	m	The more recent date.
date0	long	I	m	The less recent date.
<return>	long	O	m	One of the return codes specified in Table B.347.

B.51 Builtin Date_to_Month

Purpose

The Builtin Date_to_Month calculates the month from a given number of days since 1 January 1900.

Lexical structure

Date_to_Month(date)

The lexical elements of the Builtin Date_to_Month are shown in Table B.51.

Table B.51 – Builtin Date_to_Month

Parameter name	Type	Direction	Usage	Description
date	long	I	m	The number of days. The EDD application shall support an implicit typecast between long and DATE, so this Builtin will also accept DATE types.
<return>	int	O	m	The month.

B.52 Builtin DATE_to_string

Purpose

The Builtin DATE_to_string creates a string representation of a DATE. The ANSI C strftime function defines the structure of the format string.

Lexical structure

DATE_to_string(string, format, date)

The lexical element of the Builtin DATE_to_string is shown in Table B.52.

Table B.52 – Builtin DATE_to_string

Parameter name	Type	Direction	Usage	Description
string	DD_STRING	I	m	The string into which formatted DATE is copied.
format	DD_STRING	I	m	How the DATE is to be formatted.
date	long	I	m	The DATE to be formatted.
<return>	int	O	m	The number of character actually copied or -1 is an error occurs.

B.53 Builtin Date_To_Time

Purpose

The Builtin Date_To_Time creates a time_t from a date value. The time of day is set to 00:00:00.

Lexical Structure

Date_To_Time(date)

The lexical elements of the Builtin Date_To_Time are shown in Table B.53.

Table B.53 – Builtin Date_To_Time

Parameter name	Type	Direction	Usage	Description
date	long	I	m	The DATE value.
<return>	time_t	O	m	A time_t value.

B.54 Builtin Date_to_Year

Purpose

The Builtin Date_to_Year calculates the year from a given number of days since 1 January 1900.

Lexical structure

Date_to_Year(date)

The lexical elements of the Builtin Date_to_Year are shown in Table B.54.

Table B.54 – Builtin Date_to_Year

Parameter name	Type	Direction	Usage	Description
date	long	I	m	The number of days. The EDD application shall support an implicit typecast between long and DATE, so this Builtin will also accept DATE types.
<return>	int	O	m	The month from 1 January 1900.

B.55 Builtin days_to_DATE

Purpose

The Builtin days_to_DATE calculates a future date by adding a number of days to a DATE.

Lexical structure

days_to_DATE(days, date)

The lexical element of the Builtin days_to_DATE is shown in Table B.55.

Table B.55 – Builtin days_to_DATE

Parameter name	Type	Direction	Usage	Description
days	long	I	m	The number of days to add to the DATE.
date	long	I	m	The initial DATE.
<return>	long	O	m	One of the return codes specified in Table B.347.

B.56 Builtin DELAY

Purpose

The Builtin DELAY will display a prompt message and pause for the specified number of seconds. The prompt may contain local variable values (see Builtin put_message for lexical structure). The prompt may NOT contain device variable values. The delay time shall be a positive number.

NOTE 1 Previously displayed messages are not guaranteed to remain on the screen.

NOTE 2 Some EDD applications will clear the display upon each call to this function.

Lexical structure

DELAY(delay_time, prompt)

The lexical elements of the Builtin DELAY are shown in Table B.56.

Table B.56 – Builtin DELAY

Parameter name	Type	Direction	Usage	Description
delay_time	unsigned integer	I	m	The delay in number of seconds.
prompt	char[]	I	m	A message to be displayed.
<return>	VOID	—	—	—

B.57 Builtin delay

Purpose

The Builtin delay will display a prompt message, and pause for the specified number of seconds. The prompt may contain local and/or device VARIABLE values (see put_message for Lexical Structure). The delay time shall be a positive number.

NOTE 1 Previously displayed messages are not guaranteed to remain on the screen.

NOTE 2 Some EDD applications will clear the display upon each call to this function.

Lexical structure

delay(delay_time, prompt, global_var_ids)

The lexical elements of the Builtin delay are shown in Table B.57.

Table B.57 – Builtin delay

Parameter name	Type	Direction	Usage	Description
delay_time	unsigned integer	I	m	The delay in number of seconds.
prompt	char[]	I	m	A message to be displayed.
global_var_ids	array of int	I	m	An array which contains unique identifiers of VARIABLE instances.
<return>	VOID	—	—	—

B.58 Builtin DELAY_TIME

Purpose

The Builtin DELAY_TIME will pause for the specified number of seconds. The delay time shall be a positive number.

Lexical structure

DELAY_TIME(delay_time)

The lexical elements of the Builtin DELAY_TIME are shown in Table B.58.

Table B.58 – Builtin DELAY_TIME

Parameter name	Type	Direction	Usage	Description
delay_time	unsigned integer	I	m	The delay in number of seconds.
<return>	VOID	—	—	—

B.59 Builtin delayfor

Purpose

The Builtin delayfor prints a string on the application display and waits for the allotted amount of time to expire. VARIABLE values may be embedded in the message string with formatting information.

Lexical structure

delayfor(duration, prompt, ids, indices, id_count)

The lexical elements of the Builtin delayfor are shown in Table B.59.

Table B.59 – Builtin delayfor

Parameter name	Type	Direction	Usage	Description
duration	long	I	m	The delay in number of seconds.
prompt	char[]	I	m	A message to be displayed.
ids	array of unsigned long	I	m	The item IDs of a VARIABLE instance used in the prompt message.
indices	array of unsigned long	I	m	The member IDs or array indices of the VARIABLES used in the prompt message. Set to zero if there is no member ID or array index.
id_count	long	I	m	The number of array elements in the arrays for ids and indices. Set to zero if the arrays for ids and indices are NULL.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.60 Builtin delayfor2

Purpose

The Builtin delayfor2 prints a string on the application display and waits for the allotted amount of time to expire. VARIABLE values may be embedded in the message string with formatting information.

Lexical structure

delayfor2(duration, prompt, blk_ids, blk_nums, ids, indices, id_count)

The lexical elements of the Builtin delayfor2 are shown in Table B.60.

Table B.60 – Builtin delayfor2

Parameter name	Type	Direction	Usage	Description
duration	long	I	m	The delay in number of seconds.
prompt	char[]	I	m	A message to be displayed.
blk_ids	array of unsigned long	I	m	The item IDs of the BLOCK_A instances used in the prompt message.

Parameter name	Type	Direction	Usage	Description
blk_nums	array of unsigned long	I	m	The occurrence numbers of the BLOCK_A instances used in the prompt message.
ids	array of unsigned long	I	m	The item IDs of a VARIABLE instance used in the prompt message.
indices	array of unsigned long	I	m	The member IDs or array indices of the VARIABLES used in the prompt message. Set to zero if there is no member ID or array index.
id_count	long	I	m	The number of array elements in the arrays for ids and indices. Set to zero if the arrays for ids and indices are NULL.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.61 Builtin DICT_ID

Purpose

The Builtin DICT_ID can be used to specify a text dictionary ID for calls to get stddict_string. It converts a text dictionary name into a text dictionary ID. The Builtins processed by the EDD tool, generating the dictionary ID for the provided dictionary name. The value, which the EDD tool replaces for DICT_ID(name) is an unsigned long data type.

A text dictionary entry consists of three parts: a unique ID, a name, and a multilingual description. This function retrieves the unique ID using the text dictionary name.

Lexical structure

DICT_ID(name)

The lexical elements of the Builtin DICT_ID are shown in Table B.61.

Table B.61 – Builtin DICT_ID

Parameter name	Type	Direction	Usage	Description
name	char[]	I	m	The name of the dictionary string.
<return>	long	O	m	The dictionary ID for the named string.

B.62 Builtin dictionary_string

Purpose

The Builtin dictionary_string returns the dictionary string associated with the given name in the current language. If the string is not available in the current language, the English string is returned. If the string is not found in the dictionary the function returns 'ERROR: Dict String Not Found'. The parameter dict_string_name is the handle of the string as it appears in the dictionary.

Lexical Structure

dictionary_string(dict_string_name)

The lexical elements of the Builtin dictionary_string are shown in Table B.62.

Table B.62 – Builtin dictionary_string

Parameter name	Type	Direction	Usage	Description
dict_string_name	reference	I	m	Handle of the string as it appears in the dictionary.
<return>	DD_STRING	o	m	The dictionary string.

B.63 Builtin DiffTime

Purpose

The Builtin DiffTime calculates the difference between two calendar times.

Lexical Structure

DiffTime(t1, t0)

The lexical elements of the Builtin DiffTime are shown in Table B.63.

Table B.63 – Builtin DiffTime

Parameter name	Type	Direction	Usage	Description
t1	time_t	l	m	The more recent time.
t0	time_t	l	m	The less recent time.
<return>	double	o	m	The difference between t1 and t0 in seconds.

B.64 Builtin discard_on_exit

Purpose

The Builtin discard_on_exit sets the termination action to discard any changes made by the method to VARIABLE values stored in a caching EDD application VARIABLE table, which have not already been sent to the device. It causes the method to discard any changes to VARIABLE values in the VARIABLE table when the method ends.

When a method modifies a VARIABLE value in a caching EDD application, it is changing a resident copy of that value. This change is effective only for the life of the method. To change the value(s) in the device, the method shall send the value(s) to the device or call send_on_exit. To save the values only in the VARIABLE table (rather than send them to the device), call the Builtin save_on_exit.

After method termination, the changed values in the VARIABLE table return to the device values that existed before they were changed by the method.

All values that have been changed but not sent to the device are handled at the termination of the method by either discarding them, sending them to the device, or saving the changes in a way that survives the end of the method (but does not affect the values in the device).

This Builtin takes no action in a non-caching EDD application because all parameter values are written directly to the device.

If a method does not call discard_on_exit, save_on_exit, or send_on_exit before exiting, any values changed by the Builtin are discarded as if the Builtin discard_on_exit had been called.

Lexical structure

discard_on_exit(VOID)

The lexical element of the Builtin discard_on_exit is shown in Table B.64.

Table B.64 – Builtin discard_on_exit

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.65 Builtin DISPLAY

Purpose

The Builtin DISPLAY displays the specified message on the screen, continuously updating the dynamic variable values used in the string (see PUT_MESSAGE for syntax). The variable value will continue to be updated until operator acknowledgment is made.

Lexical Structure

DISPLAY(prompt)

The lexical elements of the Builtin DISPLAY are shown in Table B.65.

Table B.65 – Builtin DISPLAY

Parameter name	Type	Direction	Usage	Description
prompt	DD_STRING	I	m	A message to be displayed.
<return>	VOID	—	—	—

B.66 Builtin display

Purpose

The Builtin display will display the specified message on the screen, continuously updating the dynamic VARIABLE values used in the string (see put_message for Lexical Structure). The VARIABLE value will continue to be updated until operator acknowledgement is made.

NOTE 1 Previously displayed messages are not guaranteed to remain on the screen.

NOTE 2 Some EDD applications will clear the display upon each call to this function.

Lexical structure

display(prompt, global_var_ids)

The lexical elements of the Builtin display are shown in Table B.66.

Table B.66 – Builtin display

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	A message to be displayed.
global_var_ids	array of int	I	m	An array which contains unique identifiers of VARIABLE instances.
<return>	VOID	—	—	—

B.67 Builtin display_bitenum

Purpose

The Builtin display_bitenum will display a textual representation of a BIT_ENUMERATED VARIABLE and wait for acknowledgment from the user.

NOTE 1 Previously displayed messages are not guaranteed to remain on the screen.

NOTE 2 Some EDD applications will clear the display upon each call to this function.

Lexical structure

display(bitenum_variable)

The lexical elements of the Builtin display_bitenum are shown in Table B.67.

Table B.67 – Builtin display_bitenum

Parameter name	Type	Direction	Usage	Description
Bitenum_variable	Reference	I	m	The BIT_ENUMERATED VARIABLE that shall be displayed.
<return>	VOID	—	—	—

B.68 Builtin display_builtin_error**Purpose**

The Builtin display_builtin_error displays on the application display device the text associated with a specified Builtin error. The Builtin display does not return control to the method until the user acknowledges the message.

NOTE 1 This Builtin provides a means of debugging calls to Builtin routines, which are returning an error. It should never be used in a final production method, because the types of errors returned by Builtins should never be received in a fully debugged method.

NOTE 2 The Builtin error codes may be defined in a file.

Lexical structure

```
display_builtin_error(error)
```

The lexical elements of the Builtin display_builtin_error are shown in Table B.68.

Table B.68 – Builtin display_builtin_error

Parameter name	Type	Direction	Usage	Description
error	long	I	m	A code for a Builtin error.
<return>	long	O	m	One of the return codes specified in Table B.346

B.69 Builtin display_comm_error**Purpose**

The Builtin display_comm_error displays the text string associated with a communications error on the application display device.

NOTE 1 A communications error of zero is not considered a communications error.

NOTE 2 The Builtin does not return control to the method until the user acknowledges the message.

Internally, this Builtin calls the Builtin get_comm_error_string to get the associated text string and the Builtin get_acknowledgement to display the associated text string and wait for user acknowledgement.

The communications error codes are defined by the consortia. Their related strings may be defined in a text dictionary.

Lexical structure

```
display_comm_error(error)
```

The lexical elements of the Builtin display_comm_error are shown in Table B.69.

Table B.69 – Builtin display_comm_error

Parameter name	Type	Direction	Usage	Description
error	unsigned long	I	m	A code for a communication error.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.70 Builtin display_comm_status

Purpose

The Builtin display_comm_status will display the string associated with the specified value of the comm_status_value octet. The message will remain on the screen. The displayed string will remain on the screen until operator acknowledgement is made.

NOTE 1 Previously displayed messages are not guaranteed to remain on the screen.

NOTE 2 Some EDD applications will clear the display upon each call to this function.

Lexical structure

display_comm_status(comm_status_value)

The lexical elements of the Builtin display_comm_status are shown in Table B.70.

Table B.70 – Builtin display_comm_status

Parameter name	Type	Direction	Usage	Description
comm_status_value	int	l	m	A code for a communication error.
<return>	VOID	—	—	—

B.71 Builtin display_device_status

Purpose

The Builtin display_device_status will display the string associated with the specified value of the device_status octet. The displayed string will remain on the screen until operator acknowledgement is made.

NOTE 1 Previously displayed messages are not guaranteed to remain on the screen.

NOTE 2 Some EDD applications will clear the display upon each call to this function.

Lexical structure

display_device_status(device_status_value)

The lexical elements of the Builtin display_device_status are shown in Table B.71.

Table B.71 – Builtin display_device_status

Parameter name	Type	Direction	Usage	Description
device_status_value	int	l	m	A code for the device status.
<return>	VOID	—	—	—

B.72 Builtin display_dynamics

Purpose

The Builtin display_dynamics continuously displays a message on the application output device. The Builtin updates the values specified in the message until interrupted by the user. Control is not returned to the method until the user interrupts. (Application code scans for the acknowledgment; the Builtin waits for the application code to receive the user intervention, check it, and return control to the Builtin.)

VARIABLE values may be embedded in the message string with formatting information. This call can be made for device VARIABLES or local method VARIABLES. The primary purpose of the call is to display device value information that the user wants to observe over a period of time.

Lexical structure

display_dynamics(prompt, ids, indices, id_count)

The lexical elements of the Builtin display_dynamics are shown in Table B.72.

Table B.72 – Builtin display_dynamics

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	A message to be displayed.
ids	array of unsigned long	I	m	The item IDs of the VARIABLES used in the prompt message.
indices	array of unsigned long	I	m	The member IDs or array index of the VARIABLES used in the prompt message. Set to zero if there is no member ID or array index.
id_count	long	I	m	The number of array elements in the arrays for ids and indices. Set to zero if the arrays for ids and indices are NULL.
<return>	long	O	m	One of the return codes specified in Table B.346

B.73 Builtin display_dynamics2

Purpose

The Builtin display_dynamics2 continuously displays a message on the application output device. The Builtin updates the values specified in the message until interrupted by the user. Control is not returned to the method until the user interrupts. (Application code scans for the acknowledgment; the Builtin waits for the application code to receive the user intervention, check sit, and returns control to the Builtin.)

VARIABLE values may be embedded in the message string with formatting information. This call can be made for device VARIABLES or local method VARIABLES. The primary purpose of the call is to display device value information that the user wants to observe over a period of time.

Lexical structure

```
display_dynamics2(prompt, blk_ids, blk_nums, ids, indices, id_count)
```

The lexical elements of the Builtin display_dynamics2 are shown in Table B.73.

Table B.73 – Builtin display_dynamics2

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	A message to be displayed.
blk_ids	array of unsigned long	I	m	The item IDs of the BLOCK_A instances used in the prompt message.
blk_nums	array of unsigned long	I	m	The occurrence numbers of the BLOCK_A instances used in the prompt message.
ids	array of unsigned long	I	m	The item IDs of the VARIABLES used in the prompt message.
indices	array of unsigned long	I	m	The member IDs or array index of the VARIABLES used in the prompt message. Set to zero if there is no member ID or array index.
id_count	long	I	m	The number of array elements in the arrays for ids and indices. Set to zero if the arrays for ids and indices are NULL.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.74 Builtin display_message

Purpose

The Builtin display_message displays a message on the applications output device. It returns control to the method without requiring user acknowledgment.

Device, local method VARIABLE values, and labels for device VARIABLES may be embedded in the prompt string with formatting information. This call is used to inform the user of system status changes that require no action.

Lexical structure

display_message(prompt, ids, indices, id_count)

The lexical elements of the Builtin display_message are shown in Table B.74.

Table B.74 – Builtin display_message

Parameter name	Type	Direction	Usage	Description
prompt	char[]	l	m	A message to be displayed.
ids	array of unsigned long	l	m	The item IDs of the VARIABLES used in the prompt message.
indices	array of unsigned long	l	m	The member IDs or array index of the VARIABLES used in the prompt message. Set to zero if there is no member ID or array index.
id_count	long	l	m	The number of array elements in the arrays for ids and indices. Set to zero if the arrays for ids and indices are NULL.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.75 Builtin display_message2

Purpose

The Builtin display_message2 displays a message on the applications output device. It returns control to the method without requiring user acknowledgment.

Device, local method VARIABLE values, and labels for device VARIABLES may be embedded in the prompt string with formatting information. This call is used to inform the user of system status changes that require no action.

Lexical structure

display_message2(prompt, blk_ids, blk_nums, ids, indices, id_count)

The lexical elements of the Builtin display_message2 are shown in Table B.75.

Table B.75 – Builtin display_message2

Parameter name	Type	Direction	Usage	Description
prompt	char[]	l	m	A message to be displayed.
blk_ids	array of unsigned long	l	m	The item IDs of the BLOCK_A instances used in the prompt message.
blk_nums	array of unsigned long	l	m	The occurrence numbers of the BLOCK_A instances used in the prompt message.
ids	array of unsigned long	l	m	The item IDs of the VARIABLES used in the prompt message.
indices	array of unsigned	l	m	The member IDs or array index of the VARIABLES used in the prompt message. Set to zero if there is no

Parameter name	Type	Direction	Usage	Description
	long			member ID or array index.
id_count	long	I	m	The number of array elements in the arrays for ids and indices. Set to zero if the arrays for ids and indices are NULL.
<return>	long	O	m	One of the return codes specified in Table B.346

B.76 Builtin display_response_code

Purpose

The Builtin display_response_code displays the description of a response code for an item. The display Builtin does not return control to the method until the user has acknowledged the message.

Internally, this Builtin calls the Builtin get_response_code_string to get the associated text string and the Builtin get_acknowledgement to display the associated text string and wait for user acknowledgement.

A response code is a value representing an application-specific error condition. A response code value of zero is not considered an error response code. Each response code and response code type is defined in the EDD.

The parameter id and member_id refer to a parameter, which may have a response code associated with it after it was last read or written. The parameter id and member_id are specified using the Builtins ITEM_ID and MEMBER_ID and the name or reference of the VARIABLE in question.

Lexical structure

display_response_code(id, member_id, code)

The lexical elements of the Builtin display_response_code are shown in Table B.76.

Table B.76 – Builtin display_response_code

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	m	The item ID of the item which generated the response code.
member_id	unsigned long	I	m	The member ID of the item which generated the response code.
code	unsigned long	I	m	The response code as received from Builtin get_response_code.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.77 Builtin display_response_status

Purpose

The Builtin display_response_status will display the string associated with the specified value of the response_code octet. The displayed string will remain on the screen until operator acknowledgement is made.

NOTE 1 Previously displayed messages are not guaranteed to remain on the screen.

NOTE 2 Some EDD applications will clear the display upon each call to this function.

Lexical structure

display_response_status(response_code_value)

The lexical elements of the Builtin display_response_status are shown in Table B.77.

Table B.77 – Builtin display_response_status

Parameter name	Type	Direction	Usage	Description
response_code_value	int	I	m	The response code to display in a prompt.
<return>	VOID	—	—	—

B.78 Builtin display_xmtr_status

Purpose

The Builtin display_xmtr_status will display the string associated with the specified value of the indicated transmitter VARIABLE. The displayed string will remain on the screen until operator acknowledgement is made.

NOTE 1 Previously displayed messages are not guaranteed to remain on the screen.

NOTE 2 Some EDD applications will clear the display upon each call to this function.

Lexical structure

display_xmtr_status(xmtr_variable, response_code_value)

The lexical elements of the Builtin display_xmtr_status are shown in Table B.78.

Table B.78 – Builtin display_xmtr_status

Parameter name	Type	Direction	Usage	Description
xmtr_variable	int	I	m	A reference to a VARIABLE instance.
response_code_value	int	I	m	A response code value.
<return>	VOID	—	—	—

B.79 Builtin DoubleToByte

Purpose

The Builtin DoubleToByte will provide a vector of unsigned char values representing the particular elements of a double value.

Lexical structure

DoubleToByte(in, out1, out2, out3, out4, out5, out6, out7, out8)

The lexical element of the Builtin DoubleToByte is shown in Table B.79.

Table B.79 – Builtin DoubleToByte

Parameter name	Type	Direction	Usage	Description
in	double	I	m	The double value to convert.
out1	unsigned char	O	m	First element of the double value (most significant byte).
out2	unsigned char	O	m	Second element of the double value.
out3	unsigned char	O	m	Third element of the double value.
out4	unsigned char	O	m	Fourth element of the double value.
out5	unsigned char	O	m	Fifth element of the double value.
out6	unsigned char	O	m	Sixth element of the double value.
out7	unsigned char	O	m	Seventh element of the double value.

Parameter name	Type	Direction	Usage	Description
in	double	I	m	The double value to convert.
	char			
out8	unsigned char	O	m	Eighth element of the float value (least significant byte).
<return>	VOID	—	—	—

B.80 Builtin edit_device_value

Purpose

The Builtin edit_device_value displays a prompt with a device value specified by the param_id and param_member_id. The user can edit the displayed value.

NOTE A device value may reside in the VARIABLE table of a caching EDD application, and the value may not be written to the device until a method terminates or the value is explicitly sent to the device by the method.

VARIABLE values may be embedded in the message string with formatting information. The application edits and verifies the values entered for the device value. This editing is based on VARIABLE arithmetic types defined in the original EDD for each VARIABLE. The EDDL attributes associated with VARIABLES may include

- HANDLING
- DISPLAY_FORMAT
- EDIT_FORMAT
- MIN_VALUE
- MAX_VALUE

Lexical structure

```
edit_device_value(prompt, ids, indices, id_count, param_id,
param_member_id)
```

The lexical elements of the Builtin edit_device_value are shown in Table B.80.

Table B.80 – Builtin edit_device_value

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	A message to be displayed.
ids	array of unsigned long	I	m	The item IDs of the VARIABLES used in the prompt message.
indices	array of unsigned long	I	m	The member IDs or array index of the VARIABLES used in the prompt message. Set to zero if there is no member ID or array index.
id_count	long	I	m	The number of array elements in the arrays for ids and indices. Set to zero if the arrays for ids and indices are NULL.
param_id	unsigned long	I	m	The item ID of the VARIABLE to be modified.
param_member_id	unsigned long	I	m	The member ID of the VARIABLE to be modified.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.81 Builtin edit_device_value2

Purpose

The Builtin edit_device_value2 displays a prompt with a device value specified by the param_id and param_member_id. The user can edit the displayed value.

NOTE A device value may reside in the VARIABLE table of a caching EDD application, and the value may not be written to the device until a method terminates or the value is explicitly sent to the device by the method.

VARIABLE values may be embedded in the message string with formatting information. The application edits and verifies the values entered for the device value. This editing is based on VARIABLE arithmetic types defined in the original EDD for each VARIABLE. The EDDL attributes associated with VARIABLES may include:

- HANDLING
- DISPLAY_FORMAT
- EDIT_FORMAT
- MIN_VALUE
- MAX_VALUE

Lexical structure

```
edit_device_value2(prompt, blk_ids, blk_nums, ids, indices, id_count,
blk_id, blk_num, param_id, param_member_id)
```

The lexical elements of the Builtin edit_device_value2 are shown in Table B.81.

Table B.81 – Builtin edit_device_value2

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	A message to be displayed.
blk_ids	array of unsigned long	I	m	The item IDs of the BLOCK_A instances used in the prompt message.
blk_nums	array of unsigned long	I	m	The occurrence numbers of the BLOCK_A instances used in the prompt message.
ids	array of unsigned long	I	m	The item IDs of the VARIABLES used in the prompt message.
indices	array of unsigned long	I	m	The member IDs or array index of the VARIABLES used in the prompt message. Set to zero if there is no member ID or array index.
id_count	long	I	m	The number of array elements in the arrays for ids and indices. Set to zero if the arrays for ids and indices are NULL.
blk_id	unsigned long	I	m	The item ID of the BLOCK_A instance to be modified.
blk_num	unsigned long	I	m	The occurrence number of the BLOCK_A instance to be modified.
param_id	unsigned long	I	m	The item ID of the VARIABLE to be modified.
param_member_id	unsigned long	I	m	The member ID of the VARIABLE to be modified.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.82 Builtin edit_local_value

Purpose

The Builtin edit_local_value displays a prompt with the local method VARIABLE name specified by the parameter local_var. The local_var parameter is a null-terminated C string of characters. The user can edit the displayed value of the local method VARIABLE.

Variable values may be embedded in the message string with formatting information. The EDD application is responsible for preventing invalid values from being returned by performing checks based on VARIABLE type only. The method shall perform range checking on returned values.

Lexical structure

```
edit_local_value(prompt, ids, indices, id_count, local_var)
```

The lexical elements of the Builtin edit_local_value are shown in Table B.82.

Table B.82 – Builtin edit_local_value

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	A message to be displayed.
ids	array of unsigned long	I	m	The item IDs of the VARIABLES used in the prompt message.
indices	array of unsigned long	I	m	The member IDs or array index of the VARIABLES used in the prompt message. Set to zero if there is no member ID or array index.
id_count	long	I	m	The number of array elements in the arrays for ids and indices. Set to zero if the arrays for ids and indices are NULL.
local_var	char[]	I	m	The name of the local VARIABLE that may be modified.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.83 Builtin edit_local_value2

Purpose

The Builtin edit_local_value2 displays a prompt with the local method VARIABLE name specified by the parameter local_var. The local_var parameter is a null-terminated C string of characters. The user can edit the displayed value of the local method VARIABLE.

Variable values may be embedded in the message string with formatting information. The EDD application is responsible for preventing invalid values from being returned by performing checks based on VARIABLE type only. The method shall perform range checking on returned values.

Lexical structure

```
edit_local_value2(prompt, blk_ids, blk_nums, ids, indices, id_count, local_var)
```

The lexical elements of the Builtin edit_local_value2 are shown in Table B.83.

Table B.83 – Builtin edit_local_value2

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	A message to be displayed.
blk_ids	array of unsigned long	I	m	The item IDs of the BLOCK_A instances used in the prompt message.

Parameter name	Type	Direction	Usage	Description
blk_nums	array of unsigned long	I	m	The occurrence numbers of the BLOCK_A instances used in the prompt message.
ids	array of unsigned long	I	m	The item IDs of the VARIABLES used in the prompt message.
indices	array of unsigned long	I	m	The member IDs or array index of the VARIABLES used in the prompt message. Set to zero if there is no member ID or array index.
id_count	long	I	m	The number of array elements in the arrays for ids and indices. Set to zero if the arrays for ids and indices are NULL.
local_var	char[]	I	m	The name of the local VARIABLE that may be modified.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.84 Builtin exp

Purpose

The Builtin exp returns the exponential of a floating-point value.

Lexical structure

exp (x)

The lexical elements of the Builtin exp are shown in Table B.84.

Table B.84 – Builtin exp

Parameter name	Type	Direction	Usage	Description
x	double	I	m	A floating-point value.
<return>	double	O	m	The exponential value.

B.85 Builtin ext_send_command

Purpose

The Builtin ext_send_command and has the same functionality as the send_command function, except that the command status and data octets returned from the device with command 48 are also returned.

The calling function is responsible for allocating the arrays for the data to be returned. The cmd_status, and more_data_status VARIABLES are pointers to 3-octet arrays, and the more_data_info VARIABLE is a pointer to an array whose size is equal to the number of data octets that can be returned by the device in command 48 (maximum 25). If the status bit indicating that more data is available is returned (status class MORE), command 48 is automatically issued. If command 48 was not issued, the more_data_status, and more_data_info octets are set to zero.

Lexical structure

ext_send_command(cmd_number, cmd_status, more_data_status, more_data_info)

The lexical elements of the Builtin ext_send_command are shown in Table B.85.

Table B.85 – Builtin ext_send_command

Parameter name	Type	Direction	Usage	Description
cmd_number	int	I	m	The COMMAND number.
cmd_status	char[]	I	m	The COMMAND status.

Parameter name	Type	Direction	Usage	Description
more_data_status	char[]	I	m	Specifies if more status data is available.
more_data_info	char[]	I	m	Specifies if more info is available.
<return>	int	O	m	One of the return codes specified in Table B.347.

B.86 Builtin ext_send_command_trans

Purpose

The Builtin ext_send_command_trans sends the command with the specified transaction to the device. This function is to be used to send commands that have been defined with multiple transactions.

Builtin ext_send_command_trans has the same functionality as the send_command function, except that the command status and data octets returned from the device with command 48 are also returned.

The calling function is responsible for allocating the arrays for the data to be returned. The cmd_status, and more data status variables are pointers to 3-octet arrays, and the more data info variable is a pointer to an array whose size is equal to the number of data octets that can be returned by the device in command 48 (maximum 25). If command 48 was not issued, the more_data_status, and more_data_info octets are set to zero.

Lexical structure

```
ext_send_command_trans(cmd_number, transaction, cmd_status,
more_data_status, more_data_info)
```

The lexical elements of the Builtin ext_send_command_trans are shown in Table B.86.

Table B.86 – Builtin ext_send_command_trans

Parameter name	Type	Direction	Usage	Description
cmd_number	int	I	m	The COMMAND number.
transaction	int	I	m	The number of the TRANSACTION.
cmd_status	char[]	I	m	The COMMAND status.
more_data_status	char[]	I	m	Specifies if more status data is available.
more_data_info	char[]	I	m	Specifies if more info is available.
<return>	int	O	m	One of the return codes specified in Table B.347.

B.87 Builtin fail_on_all_comm_errors

Purpose

The Builtin fail_on_all_comm_errors presents the Builtins to send a return code to the method instead of aborting the method or retrying the communications request when any communications error is received.

Instead of aborting the method or attempting to retry a failing communications request when any communications error occurs, the Builtin fail_on_all_comm_errors initializes the Builtins to send a communications error code to the method.

When a communications error occurs, the Builtin performing a communications request at the time returns the error BLTIN_FAIL_COMM_ERROR. The method can call get_comm_error and subsequently get_comm_error_string to obtain additional information about the error and later display the error string, if appropriate.

Lexical structure

```
fail_on_all_comm_errors(VOID)
```


The lexical element of the Builtin fail_on_all_comm_errors is shown in Table B.87.

Table B.87 – Builtin fail_on_all_comm_errors

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.88 Builtin fail_on_all_response_codes

Purpose

The Builtin fail_on_all_response_codes presents the Builtins to return a Builtin error code to the method instead of aborting the method or retrying when any response code is received.

Instead of aborting the method or attempting to retry the failed application request, the Builtin fail_on_all_response_codes initializes the Builtins to send a return code to the method when any response code is received.

When a response code error occurs, any Builtin performing a request at the time returns the error BLTIN_FAIL_RESPONSE_CODE. The method can call get_comm_error and subsequently get_comm_error_string to obtain additional information about the error, if appropriate.

A response code is a value representing an application-specific error condition.

NOTE A response code value of zero is not considered an error response code. Each response code and response code type is defined in the EDD.

Lexical structure

fail_on_all_response_codes(VOID)

The lexical element of the Builtin fail_on_all_response_codes is shown in Table B.88.

Table B.88 – Builtin fail_on_all_response_codes

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.89 Builtin fail_on_comm_error

Purpose

The Builtin fail_on_comm_error presents the Builtins to return a return code to the method instead of aborting the method or retrying when a communication error is received.

Instead of aborting the method or attempting to retry a failed communications request, the Builtin fail_on_comm_error initializes the Builtins to return a Builtin error code to the method when a communications error occurs.

When the specified communications error occurs, any Builtin performing a communications request at the time returns the error Builtin fail_on_comm_error. The method can call get_comm_error and then get_comm_error_string to obtain additional information about the error and later display it, if appropriate.

The communications error codes are defined by the consortia. Their related strings may be defined in a text dictionary. A communications error of zero is not considered a communications error.

Lexical structure

fail_on_comm_error(error)

The lexical elements of the Builtin fail_on_comm_error are shown in Table B.89.

Table B.89 – Builtin fail_on_comm_error

Parameter name	Type	Direction	Usage	Description
error	unsigned long	I	m	A communication error code.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.90 Builtin fail_on_response_code**Purpose**

The Builtin fail_on_response_code sets the Builtins to return a Builtin error code to the method instead of aborting the method or retrying when a specified error response code is received.

Instead of aborting the method or retrying a failing request, the Builtin fail_on_response_code initializes the Builtins to return a return code to the method when a response code error occurs.

When the response code error occurs, any Builtin performing a request at the time returns the error BLTIN_FAIL_RESPONSE_CODE. The method can call get_comm_error and subsequently get_comm_error_string to obtain additional information about the error and later display it, if appropriate.

A response code is a value representing an application-specific error condition.

NOTE A response code value of zero is not considered an error response code. Each response code and response code type is defined in the EDD.

Lexical structure

```
fail_on_response_code(code)
```

The lexical elements of the Builtin fail_on_response_code are shown in Table B.90.

Table B.90 – Builtin fail_on_response_code

Parameter name	Type	Direction	Usage	Description
code	unsigned long	I	m	A response code.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.91 Builtin fassign**Purpose**

The Builtin fassign assigns a value to a VARIABLE.

Lexical structure

```
fassign(device_var, new_value)
```

The lexical elements of the Builtin fassign are shown in Table B.91.

Table B.91 – Builtin fassign

Parameter name	Type	Direction	Usage	Description
device_var	reference	I	m	The name of the VARIABLE.
new_value	double	I	m	The new value of the referenced VARIABLE.
<return>	int	O	m	One of the return codes specified in Table B.347.

B.92 Builtin fgetval

Purpose

The Builtin fgetval is specifically designed to be used in pre-read/write and post-read/write methods. This function will return the value of a device VARIABLE as it was received from the connected device. Scaling operations may then be performed on the VARIABLE value before it is stored in the EDD application.

Lexical structure

fgetval (VOID)

The lexical element of the Builtin fgetval is shown in Table B.92.

Table B.92 – Builtin fgetval

Parameter name	Type	Direction	Usage	Description
<return>	double	O	m	The value of the variable.

B.93 Builtin float_value

Purpose

The Builtin float_value returns the value of the specified VARIABLE.

Lexical structure

float_value (source_var_name)

The lexical elements of the Builtin float_value are shown in Table B.93.

Table B.93 – Builtin float_value

Parameter name	Type	Direction	Usage	Description
source_var_name	reference	I	m	The name of a VARIABLE reference.
<return>	double	O	m	The value of the variable.

B.94 Builtin FloatToByte

Purpose

The Builtin FloatToByte will provide a vector of unsigned char values representing the particular elements of a float value.

Lexical structure

FloatToByte (in, out1, out2, out3, out4)

The lexical element of the Builtin FloatToByte is shown in Table B.94.

Table B.94 – Builtin FloatToByte

Parameter name	Type	Direction	Usage	Description
in	float	I	m	The float value to convert.
out1	unsigned char	O	m	First element of the float value (most significant byte).
out2	unsigned char	O	m	Second element of the float value.
out3	unsigned char	O	m	Third element of the float value.
out4	unsigned char	O	m	Fourth element of the float value (least significant byte).

Parameter name	Type	Direction	Usage	Description
in	float	I	m	The float value to convert.
<return>	VOID	—	—	—

B.95 Builtin floor

Purpose

The Builtin floor returns the largest integral value not greater than a specified floating point value.

Lexical structure

`floor(x)`

The lexical elements of the Builtin floor are shown in Table B.95.

Table B.95 – Builtin floor

Parameter name	Type	Direction	Usage	Description
x	double	I	m	A floating point value.
<return>	double	O	m	The floor value.

B.96 Builtin fmod

Purpose

The Builtin fmod returns the floating point remainder of the division of two floating point values.

Lexical structure

`fmod(x, y)`

The lexical elements of the Builtin fmod are shown in Table B.96.

Table B.96 – Builtin fmod

Parameter name	Type	Direction	Usage	Description
x	double	I	m	The numerator.
y	double	I	m	The denominator.
<return>	double	O	m	The remainder.

B.97 Builtin fpclassify

Purpose

The Builtin fpclassify classifies a floating point value as either not a number, infinite, zero, subnormal, or normal.

NOTE This Builtin is implemented as a macro that generates a call to either `fpclassifyf` or `fpclassifyd` depending on whether its input parameter is a float or double, respectively.

Lexical Structure

`fpclassify(value)`

The lexical elements of the Builtin fpclassify are shown in Table B.97.

Table B.97 – Builtin fpclassifyd

Parameter name	Type	Direction	Usage	Description
value	double	I	m	The floating point value to be classified.
<return>	int	O	m	FP_NAN when the floating-point number x is 'Not a Number'. FP_INFINITE when the value of x is either plus or minus infinity. FP_ZERO when the value of x is positive or negative zero. FP_SUBNORMAL when the value of x is denormalized. Denormalized numbers have values that are too small (i.e., values very close to zero) to be represented otherwise. FP_NORMAL is returned for all other values of x

B.98 Builtin From_DATE_AND_TIME_VALUE

Purpose

The Builtin From_DATE_AND_TIME_VALUE creates a time_t from a DATE and a TIME_VALUE.

Lexical structure

From_DATE_AND_TIME_VALUE(date, time_value)

The lexical element of the Builtin From_DATE_AND_TIME_VALUE is shown in Table B.98.

Table B.98 – Builtin From_DATE_AND_TIME_VALUE

Parameter name	Type	Direction	Usage	Description
date	long	I	m	The DATE value.
time_value	unsigned long	I	m	The TIME_VALUE value.
<return>	time_t	O	m	The time_t that was created or -1 is an invalid parameter is detected.

B.99 Builtin From_TIME_VALUE

Purpose

The Builtin From_TIME_VALUE creates a time_t from the current date and a TIME_VALUE.

Lexical structure

From_TIME_VALUE(time_value)

The lexical element of the Builtin From_TIME_VALUE is shown in Table B.99.

Table B.99 – Builtin From_TIME_VALUE

Parameter name	Type	Direction	Usage	Description
time_value	unsigned long	I	M	The TIME_VALUE value.
<return>	time_t	O	m	The time_t that was created or -1 is an invalid parameter is detected.

B.100 Builtin fsetval

Purpose

The Builtin fsetval sets the value of a VARIABLE of type float for which a pre/post action has been designated to the specified value.

The function `fsetval` is specifically designed to be used in pre-read/write and post-read/write methods. This function will set the value of a device VARIABLE to the specified value.

Lexical structure

`fsetval(value)`

The lexical elements of the Builtin `fsetval` are shown in Table B.100.

Table B.100 – Builtin `fsetval`

Parameter name	Type	Direction	Usage	Description
value	double	I	m	The. new value for the VARIABLE.
<return>	double	O	m	The value of the variable.

B.101 Builtin `ftoa`

Purpose

The Builtin converts a floating point value into a text string.

Lexical structure

`ftoa(f)`

The lexical elements of the Builtin `ftoa` are shown in Table B.101.

Table B.101 – Builtin `ftoa`

Parameter name	Type	Direction	Usage	Description
f	float	I	m	A floating-point value to convert to a text.
<return>	char[]	O	m	The floating point value as a string.

B.102 Builtin `fvar_value`

Purpose

The Builtin `fvar_value` returns the value of the specified VARIABLE.

Lexical structure

`fvar_value(source_var)`

The lexical elements of the Builtin `fvar_value` are shown in Table B.102.

Table B.102 – Builtin `fvar_value`

Parameter name	Type	Direction	Usage	Description
source_var	reference	I	m	The name of the VARIABLE reference.
<return>	double	O	m	The value of the variable.

B.103 Builtin `get_acknowledgement`

Purpose

The Builtin `get_acknowledgement` displays the provided prompt message on the application display device and returns to the method after the user acknowledges the message. Variable values may be embedded in the message string with formatting information.

Lexical structure

`get_acknowledgement(prompt, ids, indices, id_count)`

The lexical elements of the Builtin `get_acknowledgement` are shown in Table B.103.

Table B.103 – Builtin `get_acknowledgement`

Parameter name	Type	Direction	Usage	Description
<code>prompt</code>	<code>char[]</code>	I	m	A message to be displayed.
<code>ids</code>	array of unsigned long	I	m	The item IDs of the VARIABLES used in the prompt message.
<code>indices</code>	array of unsigned long	I	m	The member IDs or array indices of the VARIABLES used in the prompt message. Set to zero if there is no member ID or array indices.
<code>id_count</code>	long	I	m	The number of array elements in the arrays for <code>ids</code> and <code>indices</code> . Set to zero if the arrays for <code>ids</code> and <code>indices</code> are NULL.
<code><return></code>	long	O	m	One of the return codes specified in Table B.346.

B.104 Builtin `get_acknowledgement2`

Purpose

The Builtin `get_acknowledgement2` displays the provided prompt message on the application display device and returns to the method after the user acknowledges the message. Variable values may be embedded in the message string with formatting information.

Lexical structure

`get_acknowledgement2(prompt, blk_ids, blk_nums, ids, indices, id_count)`

The lexical elements of the Builtin `get_acknowledgement2` are shown in Table B.104.

Table B.104 – Builtin `get_acknowledgement2`

Parameter name	Type	Direction	Usage	Description
<code>prompt</code>	<code>char[]</code>	I	m	A message to be displayed.
<code>blk_ids</code>	array of unsigned long	I	m	The item IDs of the BLOCK_A instances used in the prompt message.
<code>blk_nums</code>	array of unsigned long	I	m	The occurrence numbers of the BLOCK_A instances used in the prompt message.
<code>ids</code>	array of unsigned long	I	m	The item IDs of the VARIABLES used in the prompt message.
<code>indices</code>	array of unsigned long	I	m	The member IDs or array indices of the VARIABLES used in the prompt message. Set to zero if there is no member ID or array indices.
<code>id_count</code>	long	I	m	The number of array elements in the arrays for <code>ids</code> and <code>indices</code> . Set to zero if the arrays for <code>ids</code> and <code>indices</code> are NULL.
<code><return></code>	long	O	m	One of the return codes specified in Table B.346.

B.105 Builtin `get_block_instance_by_object_index`

Purpose

The Builtin `get_block_instance_by_object_index` returns the occurrence number of the BLOCK_A instance at the specified object index.

Lexical structure

`get_block_instance_by_object_index(obj_idx, blk_num)`

The lexical elements of the Builtin `get_block_instance_by_object_index` are shown in Table B.105.

Table B.105 – Builtin `get_block_instance_by_object_index`

Parameter name	Type	Direction	Usage	Description
obj_idx	unsigned long	I	m	The object index of the BLOCK_A instance to be returned.
blk_num	unsigned long	O	m	The occurrence number of the specified BLOCK_A instance.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.106 Builtin `get_block_instance_by_tag`

Purpose

The Builtin `get_block_instance_by_tag` returns the occurrence number of the BLOCK_A instance with the specified tag.

Lexical structure

`get_block_instance_by_tag(blk_id, blk_tag, blk_num)`

The lexical elements of the Builtin `get_block_instance_by_tag` are shown in Table B.106.

Table B.106 – Builtin `get_block_instance_by_tag`

Parameter name	Type	Direction	Usage	Description
blk_id	unsigned long	I	m	The item ID of the BLOCK_A instance to be returned.
blk_tag	char[]	I	m	The tag of the BLOCK_A instance to be returned.
blk_num	unsigned long	O	m	The occurrence number of the specified BLOCK_A instance.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.107 Builtin `get_block_instance_count`

Purpose

The Builtin `get_block_instance_count` returns the number of BLOCK_A instances that are instantiated in the device.

Lexical structure

`get_block_instance_count(blk_id, count)`

The lexical elements of the Builtin `get_block_instance_count` are shown in Table B.107.

Table B.107 – Builtin `get_block_instance_count`

Parameter name	Type	Direction	Usage	Description
blk_id	unsigned long	I	m	The item ID of the BLOCK_A instances to be counted.
count	unsigned long	O	m	The number of BLOCK_A instances of the specified type that are instantiated in the device.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.108 Builtin get_comm_error

Purpose

The Builtin get_comm_error returns the communications error code received from an application in the event of a Builtin return code of BLTIN_FAIL_COMM_ERROR.

This Builtin is used primarily for debugging purposes to determine what type of communications failures are occurring by displaying the communications error for the method writer to see.

In rare cases, the method could react to specific communications errors. However, the method should specify alternate handling of communications errors by way of the following:

- fail_on_all_comm_errors
- fail_on_comm_error
- abort_on_all_comm_errors
- ABORT_ON_COMM_ERROR

The alternate handling may cause the method to abort or immediately process the error based on the presence of a communications error.

The communications error codes are defined by the consortia. Their related strings may be defined in a text dictionary. A communications error of zero is not considered a communications error. The Builtin gets the communications error code returned from the last communications request.

Lexical structure

```
get_comm_error(VOID)
```

The lexical element of the Builtin get_comm_error is shown in Table B.108.

Table B.108 – Builtin get_comm_error

Parameter name	Type	Direction	Usage	Description
<return>	unsigned long	O	m	A communication error code.

B.109 Builtin get_comm_error_string

Purpose

The Builtin get_comm_error_string formats a descriptive string describing the communications error based on the error code that is provided. If possible, the description of the communications error is used; if the description is not available, a default communications error message with the error number is used.

Lexical structure

```
get_comm_error_string(error, string, maxlen)
```

The lexical elements of the Builtin get_comm_error_string are shown in Table B.109.

Table B.109 – Builtin get_comm_error_string

Parameter name	Type	Direction	Usage	Description
error	unsigned long	I	m	The communication error code.
string	char[]	I	m	A description of the communication error.
maxlen	long	I	m	The maximal length of string.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.110 Builtin get_date

Purpose

The Builtin `get_date` retrieves the current unscaled date value of the VARIABLE being operated on. The Builtin is called only by methods associated with a pre-edit, post-edit, post-read, or pre-write action.

The methods associated with pre-edit, post-edit, post-read, or pre-write actions are run when a VARIABLE, which has one of these actions defined as an attribute, is read or written by an application, a user method, or an edit method.

Lexical structure

```
get_date(data, size)
```

The lexical elements of the Builtin `get_date` are shown in Table B.110.

Table B.110 – Builtin get_date

Parameter name	Type	Direction	Usage	Description
data	char[]	I	m	The returned value of the VARIABLE. The structure of this output buffer is defined in this specification. This output parameters shall point to valid storage.
size	long	I	m	The returned length of the VARIABLE. This output parameter shall point to valid storage.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.111 Builtin get_date_lelem

Purpose

The Builtin `get_date_lelem` extracts a date value from a LIST.

Lexical structure

```
get_date_lelem(list_id, index, element_id, subelement_id, data, size)
```

The lexical element of the Builtin `get_date_lelem` is shown in Table B.111.

Table B.111 – Builtin get_date_lelem

Parameter name	Type	Direction	Usage	Description
list_id	unsigned long	I	m	The item id of the list.
index	int	I	m	The index of the list containing the item to be extracted.
element_id	unsigned long	I	m	The item id of the element to be extracted.
subelement_id	unsigned long	I	m	The member id of the element to be extracted.
data	char *	O	m	The buffer in which the data is to be returned.
size	int *	O	m	The size of the data returned.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.112 Builtin get_date_lelem2

Purpose

The Builtin `get_date_lelem2` extracts a date value from an embedded list.

Lexical structure

```
get_date_lelem2(list_id, index, embedded_list_id, embedded_list_index,
element_id, subelement_id, data, size)
```

The lexical element of the Builtin get_date_lelem2 is shown in Table B.112.

Table B.112 – Builtin get_date_lelem2

Parameter name	Type	Direction	Usage	Description
list_id	unsigned long	I	m	The item id of the list.
index	int	I	m	The index of the list containing the item to be extracted.
embedded_list_id	unsigned long	I	m	The item id of the embedded list.
embedded_list_index	int	I	m	The index of the embedded list containing the item to be extracted.
element_id	unsigned long	I	m	The item id of the element to be extracted.
subelement_id	unsigned long	I	m	The member id of the element to be extracted.
data	char *	O	m	The buffer in which the data is to be returned.
size	int *	O	m	The size of the data returned.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.113 Builtin get_date_value

Purpose

The Builtin get_date_value retrieves the current scaled value of the specified VARIABLE from the VARIABLE table (caching EDD application) or device and returns it.

The value is an octet buffer that may contain up to eight characters and a VARIABLE of device type date-and-time, time, or duration.

Lexical structure

```
get_date_value(id, member_id, data, size)
```

The lexical elements of the Builtin get_date_value are shown in Table B.113.

Table B.113 – Builtin get_date_value

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	m	An ID of a VARIABLE reference.
member_id	unsigned long	I	m	A member ID of a BLOCK_A, PARAMETER_LIST, RECORD, VALUE_ARRAY, COLLECTION or REFERENCE_ARRAY.
data	char[]	I	m	A buffer of max. 8 octet for the result of the Builtin.
size	long	I	m	The size of the used buffer.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.114 Builtin get_date_value2

Purpose

The Builtin get_date_value2 retrieves the current scaled value of the specified VARIABLE from the VARIABLE table (caching EDD application) or device and returns it.

The value is a octet buffer that may contain up to eight characters and contain a VARIABLE of device type date-and-time, time, or duration.

Lexical structure

```
get_date_value2(blk_id, blk_num, id, member_id, data, size)
```

The lexical elements of the Builtin get_date_value2 are shown in Table B.114.

Table B.114 – Builtin get_date_value2

Parameter name	Type	Direction	Usage	Description
blk_id	unsigned long	I	m	The item ID of the BLOCK_A instance to access.
blk_num	unsigned long	I	m	The occurrence number of the BLOCK_A instance to access.
id	unsigned long	I	m	An ID of a VARIABLE reference.
member_id	unsigned long	I	m	A member ID of a BLOCK_A, PARAMETER_LIST, RECORD, VALUE_ARRAY, COLLECTION or REFERENCE_ARRAY.
data	char[]	I	m	A buffer of max. 8 octet for the result of the Builtin.
size	long	I	m	The size of the used buffer.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.115 Builtin GET_DD_REVISION

Purpose

The Builtin GET_DD_REVISION gets the the DD_REVISION from the currently used device description as stated in the identification of the device description with the keyword DD_REVISION.

Lexical Structure

```
GET_DD_REVISION (VOID)
```

The lexical elements of the Builtin GET_DD_REVISION are shown in Table B.115.

Table B.115 – Builtin GET_DD_REVISION

Parameter name	Type	Direction	Usage	Description
<return>	int	o	m	The current DD_REVISION.

B.116 Builtin get_dds_error

Purpose

The Builtin get_dds_error provides a means of debugging calls to Builtin routines, which are returning an error. It should never be used in a final production method, because the types of errors that are returned with get_dds_error should never be received in a fully debugged method.

A method may determine the actual EDD error received after the error return BLTIN_DDS_ERROR is received by calling this Builtin. Optionally, an allocated character pointer with its length may be passed to get the text from the error.

The EDD error and optional description returned are for the most recent EDD error.

Returns the EDD error generated by an earlier call, and optionally returns a text string describing the error.

Lexical structure

`get_dds_error(string, maxlen)`

The lexical elements of the Builtin `get_dds_error` are shown in Table B.116.

Table B.116 – Builtin `get_dds_error`

Parameter name	Type	Direction	Usage	Description
string	char[]	I	m	A pointer to a text description of the error.
maxlen	long	I	m	The maximum length of the optional output buffer (string) in octets. This parameter is meaningful only if a parameter string has been allocated for the error message.
<return>	long	O	m	An error code generated by an earlier Builtin call.

B.117 Builtin `GET_DEV_VAR_VALUE`

Purpose

The Builtin `GET_DEV_VAR_VALUE` will display the specified prompt message, and allow the user to edit the value of a device VARIABLE. The edited copy of the device VARIABLE value will be updated when the new value is entered, but will not be sent to the device. This shall be done explicitly using one of the send command functions.

The prompt may NOT contain embedded local and/or device VARIABLE values.

NOTE 1 Previously displayed messages are not guaranteed to remain on the screen.

NOTE 2 Some EDD applications will clear the display upon each call to this function.

Lexical structure

`GET_DEV_VAR_VALUE(prompt, device_var_name)`

The lexical elements of the Builtin `GET_DEV_VAR_VALUE` are shown in Table B.117.

Table B.117 – Builtin `GET_DEV_VAR_VALUE`

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	A message to be displayed.
device_var_name	reference	I	m	The referenced name of the VARIABLE, which can be edited.
<return>	int	O	m	One of the return codes specified in Table B.347.

B.118 Builtin `get_dev_var_value`

Purpose

The Builtin `get_dev_var_value` will display the specified prompt message and allow the user to edit the value of a device VARIABLE. The edited copy of the device VARIABLE value will be updated when the new value is entered but will not be sent to the device. This shall be done explicitly using one of the send command functions.

The prompt may contain embedded local and/or device VARIABLE values (see Builtin `PUT_MESSAGE`).

NOTE 1 Previously displayed messages are not guaranteed to remain on the screen.

NOTE 2 Some EDD applications will clear the display upon each call to this function.

Lexical structure

`get_dev_var_value(prompt, global_var_ids, device_var_name)`

The lexical elements of the Builtin `get_dev_var_value` are shown in Table B.118.

Table B.118 – Builtin get_dev_var_value

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	A message to be displayed.
global_var_ids	array of int	I	m	An array which contains unique identifiers of VARIABLE instance.
device_var_name	reference	I	m	The name of the VARIABLE, which can be edited.
<return>	int	O	m	One of the return codes specified in Table B.347.

B.119 Builtin GET_DEVICE_REVISION**Purpose**

The Builtin GET_DEVICE_REVISION gets the DEVICE_REVISION from the currently used device description as stated in the identification of the device description with the keyword DEVICE_REVISION.

Lexical Structure

GET_DEVICE_REVISION (VOID)

The lexical elements of the Builtin GET_DEVICE_REVISION are shown in Table B.119.

Table B.119 – Builtin GET_DEVICE_REVISION

Parameter name	Type	Direction	Usage	Description
<return>	int	o	m	The current DEVICE_REVISION.

B.120 Builtin GET_DEVICE_TYPE**Purpose**

The Builtin GET_DEVICE_TYPE gets the numerical equivalent of the DEVICE_TYPE from the currently used device description as stated in the identification of the device description with the keyword DEVICE_TYPE.

Lexical Structure

GET_DEVICE_TYPE (VOID)

The lexical elements of the Builtin GET_DEVICE_TYPE are shown in Table B.120.

Table B.120 – Builtin GET_DEVICE_TYPE

Parameter name	Type	Direction	Usage	Description
<return>	int	o	m	The current DEVICE_TYPE.

B.121 Builtin get_dictionary_string**Purpose**

The Builtin get_dictionary_string will retrieve the text dictionary string associated with the given name in the current language. If the string is not available in the current language, the English string will be retrieved. If the string is not defined in either language, an error condition occurs, and the function will return FALSE. If the string is longer than the max_str_len, the string will be truncated. The parameter dict_string_name is the name of the string as it appears in the text dictionary.

Lexical structure

get_dictionary_string(dict_string_name, string, max_str_len)

The lexical elements of the Builtin `get_dictionary_string` are shown in Table B.121.

Table B.121 – Builtin `get_dictionary_string`

Parameter name	Type	Direction	Usage	Description
<code>dict_string_name</code>	reference	I	m	The name of a dictionary entry.
<code>string</code>	char[]	I	m	A buffer for the result string.
<code>max_str_len</code>	int	I	m	The maximum length of the buffer.
<return>	int	O	m	One of the return codes specified in Table B.347.

B.122 Builtin `get_double`

Purpose

The Builtin `get_double` retrieves the current unscaled double value of the VARIABLE being operated on.

Gets the current value of the double VARIABLE as if it was directly read from a device.

The Builtin `get_double` is called only by methods associated with a pre-edit, post-edit, post-read or pre-write action.

The methods associated with pre-edit, post-edit, post-read, or pre-write actions are run when a VARIABLE, which has one of these actions defined as an attribute, is read by an application or another method.

Lexical structure

`get_double(value)`

The lexical elements of the Builtin `get_double` are shown in Table B.122.

Table B.122 – Builtin `get_double`

Parameter name	Type	Direction	Usage	Description
<code>value</code>	double	O	m	The returned value of the VARIABLE.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.123 Builtin `get_double_lelem`

Purpose

The Builtin `get_double_lelem` extracts a double value from a LIST.

Lexical structure

`get_double_lelem(list_id, index, element_id, subelement_id)`

The lexical element of the Builtin `get_double_lelem` is shown in Table B.123.

Table B.123 – Builtin `get_double_lelem`

Parameter name	Type	Direction	Usage	Description
<code>list_id</code>	unsigned long	I	m	The item id of the list.
<code>index</code>	int	I	m	The index of the list containing the item to be extracted.
<code>element_id</code>	unsigned long	I	m	The item id of the element to be extracted.
<code>subelement_id</code>	unsigned long	I	m	The member id of the element to be extracted.

Parameter name	Type	Direction	Usage	Description
list_id	unsigned long	I	m	The item id of the list.
<return>	double	O	m	The specified value.

B.124 Builtin get_double_lelem2

Purpose

The Builtin get_double_lelem2 extracts a double value from an embedded list.

Lexical structure

```
get_double_lelem2(list_id, index, embedded_list_id, embedded_list_index,
element_id, subelement_id)
```

The lexical element of the Builtin get_double_lelem2 is shown in Table B.124.

Table B.124 – Builtin get_double_lelem2

Parameter name	Type	Direction	Usage	Description
list_id	unsigned long	I	m	The item id of the list.
index	int	I	m	The index of the list containing the item to be extracted.
embedded_list_id	unsigned long	I	m	The item id of the embedded list.
embedded_list_index	int	I	m	The index of the embedded list containing the item to be extracted.
element_id	unsigned long	I	m	The item id of the element to be extracted.
subelement_id	unsigned long	I	m	The member id of the element to be extracted.
<return>	double	O	m	The specified value.

B.125 Builtin get_double_value

Purpose

The Builtin get_double_value retrieves the current scaled value of the specified VARIABLE from the VARIABLE table (caching EDD application) or device and returns it.

The parameters id and member_id are specified using the Builtins ITEM_ID and MEMBER_ID and the name or reference of the variable.

The VARIABLE shall have a data type of double and be a valid variable in the VARIABLE table (caching EDD application) or device.

Lexical structure

```
get_double_value(id, member_id, value)
```

The lexical elements of the Builtin get_double_value are shown in Table B.125.

Table B.125 – Builtin get_double_value

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	m	The Item ID of the VARIABLE to access.
member_id	unsigned long	I	m	The member ID of the VARIABLE to access.

Parameter name	Type	Direction	Usage	Description
value	double	O	m	The returned value of the VARIABLE.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.126 Builtin get_double_value2

Purpose

The Builtin get_double_value2 retrieves the current scaled value of the specified VARIABLE from the VARIABLE table (caching EDD application) or device and returns it.

The parameters id and member_id are specified using the Builtins ITEM_ID and MEMBER_ID and the name or reference of the variable.

The VARIABLE shall have a data type of double and be a valid variable in the VARIABLE table (caching EDD application) or device.

Lexical structure

get_double_value2(blk_id, blk_num, id, member_id, value)

The lexical elements of the Builtin get_double_value2 are shown in Table B.126.

Table B.126 – Builtin get_double_value2

Parameter name	Type	Direction	Usage	Description
blk_id	unsigned long	I	m	The item ID of the BLOCK_A instance to access.
blk_num	unsigned long	I	m	The occurrence number of the BLOCK_A instance to access.
id	unsigned long	I	m	The item ID of the VARIABLE to access.
member_id	unsigned long	I	m	The member ID of the VARIABLE to access.
value	double	O	m	The returned value of the VARIABLE.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.127 Builtin get_enum_string

Purpose

The Builtin get_enum_string returns the string associated with the value of an enumerated or bit enumerated variable.

Lexical structure

get_enum_string(variable, value, string)

The lexical elements of the Builtin get_enum_string are shown in Table B.127.

Table B.127 – Builtin get_enum_string

Parameter name	Type	Direction	Usage	Description
variable	DD_ITEM	I	m	An enumerated or bit enumerated variable.
value	int	I	m	A value of the variable.
string	DD_STRING	O	m	String associated with the value of the variable.
<return>	int	O	m	One of the return codes specified in Table B.346

B.128 Builtin get_float

Purpose

The Builtin get_float retrieves the current unscaled floating-point value of the VARIABLE being operated on.

Gets the current value of an un-scaled floating-point VARIABLE as if it was directly read from a device.

The Builtin get_float can be called only by methods associated with a pre-edit, post-edit, post-read, or pre-write action. The methods associated with pre-edit, post-edit, post-read, or pre-write actions are run when a VARIABLE, which has one of these actions defined as an attribute, is read by an application, a user method, or an edit method.

Lexical structure

get_float(value)

The lexical elements of the Builtin get_float are shown in Table B.128.

Table B.128 – Builtin get_float

Parameter name	Type	Direction	Usage	Description
value	float	O	m	The returned value of the VARIABLE.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.129 Builtin get_float_lelem

Purpose

The Builtin get_float_lelem extracts a float value from a LIST.

Lexical structure

get_float_lelem(list_id, index, element_id, subelement_id)

The lexical element of the Builtin get_float_lelem is shown in Table B.129.

Table B.129 – Builtin get_float_lelem

Parameter name	Type	Direction	Usage	Description
list_id	unsigned long	I	m	The item id of the list.
index	int	I	m	The index of the list containing the item to be extracted.
element_id	unsigned long	I	m	The item id of the element to be extracted.
subelement_id	unsigned long	I	m	The member id of the element to be extracted.
<return>	float	O	m	The specified value.

B.130 Builtin get_float_lelem2

Purpose

The Builtin get_float_lelem2 extracts a float value from an embedded list.

Lexical structure

get_float_lelem2(list_id, index, embedded_list_id, embedded_list_index, element_id, subelement_id)

The lexical element of the Builtin get_float_lelem2 is shown in Table B.130.

Table B.130 – Builtin get_float_lelem2

Parameter name	Type	Direction	Usage	Description
list_id	unsigned long	I	m	The item id of the list.
index	int	I	m	The index of the list containing the item to be extracted.
embedded_list_id	unsigned long	I	m	The item id of the embedded list.
embedded_list_index	int	I	m	The index of the embedded list containing the item to be extracted.
element_id	unsigned long	I	m	The item id of the element to be extracted.
subelement_id	unsigned long	I	m	The member id of the element to be extracted.
<return>	float	O	m	The specified value.

B.131 Builtin get_float_value**Purpose**

The Builtin get_float_value retrieves the current scaled value of the specified VARIABLE from the VARIABLE table (caching EDD application) or device and returns it.

The parameters id and member id are specified using the Builtins ITEM_ID and MEMBER_ID and the name or reference of the VARIABLE.

Lexical structure

```
get_float_value(id, member_id, value)
```

The lexical elements of the Builtin get_float_value are shown in Table B.131.

Table B.131 – Builtin get_float_value

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	m	The item ID of the VARIABLE to access.
member_id	unsigned long	I	m	The member ID of the VARIABLE to access.
value	float	O	m	The returned value of the VARIABLE.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.132 Builtin get_float_value2**Purpose**

The Builtin get_float_value2 retrieves the current scaled value of the specified VARIABLE from the VARIABLE table (caching EDD application) or device and returns it.

The parameters id and member id are specified using the Builtins ITEM_ID and MEMBER_ID and the name or reference of the VARIABLE.

Lexical structure

```
get_float_value2(id, member_id, value)
```

The lexical elements of the Builtin get_float_value2 are shown in Table B.132.

Table B.132 – Builtin get_float_value2

Parameter name	Type	Direction	Usage	Description
blk_id	unsigned long	I	m	The item ID of the BLOCK_A instance to access.
blk_num	unsigned long	I	m	The occurrence number of the BLOCK_A instance to access.
id	unsigned long	I	m	The item ID of the VARIABLE to access.
member_id	unsigned long	I	m	The member ID of the VARIABLE to access.
value	float	O	m	The returned value of the VARIABLE.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.133 Builtin GET_LOCAL_VAR_VALUE**Purpose**

The Builtin GET_LOCAL_VAR_VALUE will display the specified prompt message and allow the user to edit the value of a local VARIABLE.

The prompt may NOT contain embedded local and/or device VARIABLE values.

NOTE 1 Previously displayed messages are not guaranteed to remain on the screen.

NOTE 2 Some EDD applications will clear the display upon each call to this function.

Lexical structure

GET_LOCAL_VAR_VALUE(prompt, local_var_name)

The lexical elements of the Builtin GET_LOCAL_VAR_VALUE are shown in Table B.133.

Table B.133 – Builtin GET_LOCAL_VAR_VALUE

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	A message to be displayed.
local_var_name	reference	I	m	The name of the local VARIABLE to edit.
<return>	int	O	m	One of the return codes specified in Table B.347.

B.134 Builtin get_local_var_value**Purpose**

The Builtin get_local_var_value will display the specified prompt message and allow the user to edit the value of a local VARIABLE.

The prompt may contain embedded local and/or device VARIABLE values.

NOTE 1 Previously displayed messages are not guaranteed to remain on the screen.

NOTE 2 Some EDD applications will clear the display upon each call to this function.

Lexical structure

get_local_var_value(prompt, global_var_ids, local_var_name)

The lexical elements of the Builtin get_local_var_value are shown in Table B.134.

Table B.134 – Builtin get_local_var_value

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	A message to be displayed.
global_var_ids	array of int	I	o	An array which contains unique identifiers of VARIABLE instance.
local_var_name	reference d	I	o	The name of the local VARIABLE to edit.
<return>	int	O	m	One of the return codes specified in Table B.347.

B.135 Builtin GET_MANUFACTURER

Purpose

The Builtin GET_MANUFACTURER gets the numerical equivalent of the MANUFACTURER from the currently used device description as stated in the identification of the device description with the keyword MANUFACTURER.

Lexical Structure

GET_MANUFACTURER (VOID)

The lexical elements of the Builtin GET_MANUFACTURER are shown in Table B.135.

Table B.135 – Builtin GET_MANUFACTURER

Parameter name	Type	Direction	Usage	Description
<return>	int	o	m	The current MANUFACTURER.

B.136 Builtin get_more_status

Purpose

The Builtin get_more_status will issue a command 48 to the device and return the response code, communications status, and command status octets in the status array provided. It will also return any data octets returned in the more_data_info array. This function will process the status/data octets in the same manner as the send command.

The calling function is responsible for allocating the arrays for the data to be returned. The parameter more_data_status is a 3-octet array, and the more_data_info is an array whose size is equal to the number of data octets that can be returned by the device in command 48 (maximum 25).

Lexical structure

get_more_status(more_data_status, more_data_info)

The lexical elements of the Builtin get_more_status are shown in Table B.136.

Table B.136 – Builtin get_more_status

Parameter name	Type	Direction	Usage	Description
more_data_status	char[]	I	m	Specifies if more info is available.
more_data_info	char[]	I	m	Specifies if more info is available.
<return>	int	O	m	One of the return codes specified in Table B.347.

B.137 Builtin get_resolve_status

Purpose

The Builtin `get_resolve_status` returns the error number of the most recent “`get_resolve_`” built-ins. The reference resolution Builtins, those starting with “`get_resolve_`”, resolve the possibly dynamic ID of an element of a `BLOCK_A`, `PARAMETER`, `PARAMETER_LIST`, `VALUE_ARRAY`, `REFERENCE_ARRAY`, `COLLECTION`, or `RECORD`. If any of these resolve reference Builtins encounters an error, it returns a zero for the ID.

If the calling method wants to know what the error was, then it calls the Builtin `get_resolve_status` to get the error number. This Builtins used for debugging purposes when creating methods.

Returns the codes for errors generated by Builtin `resolve_array_ref`, `resolve_block_ref`, `resolve_parm_ref`, `resolve_param_list_ref` or `resolve_record_ref`.

Lexical structure

```
get_resolve_status(VOID)
```

The lexical element of the Builtin `get_resolve_status` is shown in Table B.137.

Table B.137 – Builtin get_resolve_status

Parameter name	Type	Direction	Usage	Description
<return>	unsigned long	O	m	One of the return codes specified in Table B.346.

B.138 Builtin get_response_code

Purpose

The Builtin `get_response_code` returns the response code received from a device in the event of a Builtin communications call returning the error Builtin `fail_on_response_code`. It also returns the value reference (`err_id`, `err_member_id`) information associated with the original request. The original request is the last Builtin call that caused communication.

A response code is a value representing an application-specific error condition.

NOTE A response code value of zero is not considered an error response code. Each response code and response code type is defined in the EDD.

Gets the response code returned on the last communications request, and the item ID and member ID for the value being accessed when the response code was returned.

Lexical structure

```
get_response_code(resp_code, err_id, err_member_id)
```

The lexical elements of the Builtin `get_response_code` are shown in Table B.138.

Table B.138 – Builtin get_response_code

Parameter name	Type	Direction	Usage	Description
resp_code	unsigned long	I	m	The number of the returned response code.
err_id	unsigned long	O	m	The returned item ID of the item generating the response code.
err_member_id	unsigned long	O	m	The returned member ID of the item causing a response code.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.139 Builtin get_response_code_string

Purpose

The Builtin `get_response_code_string` searches the response codes of the specified VARIABLE and returns the description of the specified response code. If the response code is not found, and if the item is a member of a RECORD or an element of an REFERENCE_ARRAY, then the RECORD or REFERENCE_ARRAY is checked for the specified code.

The method developer calls this Builtin if the description of a response code is unknown and shall be displayed. Otherwise, if the response code is known, the method can determine what to do in the presence of certain errors. Also non-standard error descriptions can be displayed by an application instead of the standard response codes.

Before calling this Builtin, the method shall use the Builtin `get_response_code` to get the item_ID and member_ID for the item with a response code. The parameters `id` and `member_id` are specified using the Builtins ITEM_ID and MEMBER_ID and the name or reference of the VARIABLE.

Lexical structure

```
get_response_code_string(id, member_id, code, string, maxlen)
```

The lexical elements of the Builtin `get_response_code_string` are shown in Table B.139.

Table B.139 – Builtin get_response_code_string

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	m	The item ID of the item to find.
member_id	unsigned long	I	m	The member ID of the item to find.
code	unsigned long	I	m	The response code for which the descriptive string is to be found.
string	char*	O	m	The buffer for the returned response code description.
maxlen	long	I	m	The maximum length of the buffer.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.140 Builtin get_rspcode_string

Purpose

The Builtin `get_rspcode_string` will return the response code string for the specified command and response code. If the string is longer than the maximum length defined in `response_string_length`, the string is truncated. The response code specified shall be valid for the indicated command.

Lexical structure

```
get_rspcode_string(cmd_number, code, string, length)
```

The lexical elements of the Builtin `get_rspcode_string` are shown in Table B.140.

Table B.140 – Builtin get_rspcode_string

Parameter name	Type	Direction	Usage	Description
cmd_number	int	I	m	The command number.
code	int	I	m	The response code.
string	DD_STRING	O	m	The buffer for the response code string.

Parameter name	Type	Direction	Usage	Description
length	int	I		The maximum length of the string buffer.
<return>	int	O	m	One of the return codes specified in Table B.347.

B.141 Builtin get_rspcode_string_by_id

Purpose

The Builtin `get_rspcode_string_by_id` returns a string representation of a given `response_code` (returned by `ReadCommand`, `WriteCommand`).

Lexical structure

`get_rspcode_string_by_id(command, response_code, result)`

The lexical elements of the Builtin `get_rspcode_string_by_id` are shown in Table B.141.

Table B.141 – Builtin `getrsp_code_string_by_id`

Parameter name	Type	Direction	Usage	Description
command	reference	I	m	The command from which the <code>response_code</code> was delivered.
response_code	int	I	m	The <code>response_code</code> value (as returned from <code>ReadCommand</code> or <code>WriteCommand</code>).
result	DD_STRING	O	m	String representation of the given <code>response_code</code> .
<return>	int	O	m	One of the return codes specified in Table B.347.

B.142 Builtin get_signed

Purpose

The Builtin `get_signed` retrieves the current unscaled signed value of the VARIABLE being operated on.

Gets the current value of a signed VARIABLE as if it was directly read from a device.

The Builtin `get_signed` can be called only by methods associated with a pre-edit, post-edit, post-read, or pre-write action. The methods associated with pre-edit, post-edit, post-read, or pre-write actions are run when a VARIABLE, which has one of these actions defined as an attribute, is read by an application, a user method, or an edit method.

Lexical structure

`get_signed(value)`

The lexical elements of the Builtin `get_signed` are shown in Table B.142.

Table B.142 – Builtin `get_signed`

Parameter name	Type	Direction	Usage	Description
value	long	I	m	The returned value of the signed VARIABLE.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.143 Builtin get_signed_ilem

Purpose

The Builtin `get_signed_ilem` extracts a signed value from a LIST.

Lexical structure

`get_signed_lelem(list_id, index, element_id, subelement_id)`

The lexical element of the Builtin `get_signed_lelem` is shown in Table B.143.

Table B.143 – Builtin `get_signed_lelem`

Parameter name	Type	Direction	Usage	Description
list_id	unsigned long	I	m	The item id of the list.
index	int	I	m	The index of the list containing the item to be extracted.
element_id	unsigned long	I	m	The item id of the element to be extracted.
subelement_id	unsigned long	I	m	The member id of the element to be extracted.
<return>	long	O	m	The specified value.

B.144 Builtin `get_signed_lelem2`

Purpose

The Builtin `get_signed_lelem2` extracts a signed value from an embedded list.

Lexical structure

`get_signed_lelem2(list_id, index, embedded_list_id, embedded_list_index, element_id, subelement_id)`

The lexical element of the Builtin `get_signed_lelem2` is shown in Table B.144.

Table B.144 – Builtin `get_signed_lelem2`

Parameter name	Type	Direction	Usage	Description
list_id	unsigned long	I	m	The item id of the list.
index	int	I	m	The index of the list containing the item to be extracted.
embedded_list_id	unsigned long	I	m	The item id of the embedded list.
embedded_list_index	int	I	m	The index of the embedded list containing the item to be extracted.
element_id	unsigned long	I	m	The item id of the element to be extracted.
subelement_id	unsigned long	I	m	The member id of the element to be extracted.
<return>	long	O	m	The specified value.

B.145 Builtin `get_signed_value`

Purpose

The Builtin `get_signed_value` retrieves the current scaled value of the specified VARIABLE from the VARIABLE table (caching EDD application) or device and returns it.

The parameters id and member id are specified using the Builtins ITEM_ID and MEMBER_ID and the name or reference of the VARIABLE. The VARIABLE shall have a data type of signed integer and shall be a valid VARIABLE in the VARIABLE table or device.

Lexical structure

`get_signed_value(id, member_id, value)`

The lexical elements of the Builtin `get_signed_value` are shown in Table B.145.

Table B.145 – Builtin `get_signed_value`

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	m	The item ID of the VARIABLE to access.
member_id	unsigned long	I	m	The member ID of the VARIABLE to access.
value	long	O	m	The returned value of the signed VARIABLE.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.146 Builtin `get_signed_value2`

Purpose

The Builtin `get_signed_value2` retrieves the current scaled value of the specified VARIABLE from the VARIABLE table (caching EDD application) or device and returns it.

The parameters `id` and `member id` are specified using the Builtins `ITEM_ID` and `MEMBER_ID` and the name or reference of the VARIABLE. The VARIABLE shall have a data type of signed integer and shall be a valid VARIABLE in the VARIABLE table or device.

Lexical structure

```
get_signed_value2(blk_id, blk_num, id, member_id, value)
```

The lexical elements of the Builtin `get_signed_value2` are shown in Table B.146.

Table B.146 – Builtin `get_signed_value2`

Parameter name	Type	Direction	Usage	Description
blk_id	unsigned long	I	m	The item ID of the BLOCK_A instance to access.
blk_num	unsigned long	I	m	The occurrence number of the BLOCK_A instance to access.
id	unsigned long	I	m	The item ID of the VARIABLE to access.
member_id	unsigned long	I	m	The member ID of the VARIABLE to access.
value	long	O	m	The returned value of the signed VARIABLE.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.147 Builtin `get_status_code_string`

Purpose

The Builtin `get_status_code_string` will return status code string for the specified device VARIABLE and status code value. If the string is longer than the maximum length defined in `status_string_length`, the string is truncated. The status code value shall be valid for the specified device VARIABLE.

Lexical structure

```
get_status_code_string(variable_name, status_code, status_string, status_string_length)
```

The lexical elements of the Builtin `get_status_code_string` are shown in Table B.147.

Table B.147 – Builtin get_status_code_string

Parameter name	Type	Direction	Usage	Description
variable_name	char[]	I	m	The name of the VARIABLE to access.
status_code	int	I	m	The value of interesting status code.
status_string	char *	O	m	The buffer of the returned value of the signed VARIABLE.
status_string_length	int	I	m	The maximum size of the buffer.
<return>	VOID	—	—	—

B.148 Builtin get_status_string

Purpose

The Builtin get_status_string searches the members of the referenced ENUMERATED or BIT_ENUMERATED VARIABLE and returns the description of the member matching the status in the given buffer.

The Builtin gets the string associated with an ~, ENUMERATED or BIT_ENUMERATED value. The most common use of ENUMERATED and BIT_ENUMERATED VARIABLES is to get a string associated with a status bit defined in the device for a particular status. The strings associated with ENUMERATED and BIT_ENUMERATED VARIABLES are defined in the EDDL.

The parameters id and member_id are specified using the Builtins ITEM_ID and MEMBER_ID and the name or reference of the VARIABLE in question.

Gets the description of a specific status in a specific device.

Lexical structure

```
get_status_string(id, member_id, status, string, maxlen)
```

The lexical elements of the Builtin get_status_string are shown in Table B.148.

Table B.148 – Builtin get_status_string

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	m	The item ID of the VARIABLE to access.
member_id	unsigned long	I	m	The member ID of the VARIABLE to access.
status	unsigned long	I	m	The value to search.
string	char*	O	m	The buffer for the returned string.
maxlen	long	I	m	The maximum size of the buffer.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.149 Builtin get_stddict_string

Purpose

The Builtin get_stddict_string takes a text dictionary identifier and returns the corresponding text string in the supplied string buffer. The returned string is in the correct language for the application.

A text dictionary entry is composed of three parts: a unique id, a name, and a multilingual description. This Builtin returns the multilingual description portion of the entry. The Builtin strings are defined in the standard.dct file.

Lexical structure

```
get_stddict_string(id, string, maxlen)
```

The lexical elements of the Builtin `get_stddict_string` are shown in Table B.149.

Table B.149 – Builtin `get_stddict_string`

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	m	The text dictionary identifier.
string	char*	O	m	The returned buffer for the string.
maxlen	long	I	m	The maximum size of the buffer.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.150 Builtin `get_string`**Purpose**

The Builtin `get_string` retrieves the current, unscaled, string value of the VARIABLE being operated on.

Gets the current value of the string VARIABLE as if it was directly read from a device. The VARIABLE shall be a valid parameter. The VARIABLE shall have a data type of char and be an octet buffer which can contain various non-ANSI data types such as:

- ASCII
- PASSWORD
- EUC (Extended Unit Code)
- BITSTRING

Device VARIABLES of these data types shall be handled by the method because they are not supported by ANSI-C. The Builtin `get_string` is called only by methods associated with a pre-edit, post-edit, post-read, or pre-write action. The method associated with pre-edit, post-edit, post-read, or post-write actions are run when a VARIABLE, which has one of these actions defined as an attribute, is read by an application, a user method, or an edit method.

Lexical structure

```
get_string(string, len)
```

The lexical elements of the Builtin `get_string` are shown in Table B.150.

Table B.150 – Builtin `get_string`

Parameter name	Type	Direction	Usage	Description
string	char*	O	m	The buffer for the returned string.
len	long	I	m	The maximum size of the string. The returned length of the string.
<return>	int	O	m	One of the return codes specified in Table B.346.

B.151 Builtin `get_string_lelem`**Purpose**

The Builtin `get_string_lelem` extracts a string from a LIST.

Lexical structure

```
get_string_lelem(list_id, index, element_id, subelement_id, string, length)
```

The lexical element of the Builtin `get_string_lelem` is shown in Table B.151.

Table B.151 – Builtin get_string_lelem

Parameter name	Type	Direction	Usage	Description
list_id	unsigned long	I	m	The item id of the list.
index	int	I	m	The index of the list containing the item to be extracted.
element_id	unsigned long	I	m	The item id of the element to be extracted.
subelement_id	unsigned long	I	m	The member id of the element to be extracted.
string	char *	O	m	The buffer in which the data is to be returned.
length	int *	O	m	The size of the data returned.
<return>	long	O	m	The specified value.

B.152 Builtin get_string_lelem2**Purpose**

The Builtin get_string_lelem2 extracts a string from an embedded list.

Lexical structure

```
get_string_lelem2(list_id, index, embedded_list_id, embedded_list_index,
element_id, subelement_id, string, length)
```

The lexical element of the Builtin get_string_lelem2 is shown in Table B.152.

Table B.152 – Builtin get_string_lelem2

Parameter name	Type	Direction	Usage	Description
list_id	unsigned long	I	m	The item id of the list.
index	int	I	m	The index of the list containing the item to be extracted.
embedded_list_id	unsigned long	I	m	The item id of the embedded list.
embedded_list_index	int	I	m	The index of the embedded list containing the item to be extracted.
element_id	unsigned long	I	m	The item id of the element to be extracted.
subelement_id	unsigned long	I	m	The member id of the element to be extracted.
string	char *	O	m	The buffer in which the data is to be returned.
length	int *	O	m	The size of the data returned.
<return>	long	O	m	The specified value.

B.153 Builtin get_string_value**Purpose**

The Builtin get_string_value retrieves the current scaled value of the specified VARIABLE from the VARIABLE table (caching EDD application) or device and returns it.

The parameters id and member id are specified using the Builtins ITEM_ID and MEMBER_ID and the name or reference of the VARIABLE in question.

Lexical structure

```
get_string_value(id, member_id, string, len)
```

The lexical elements of the Builtin `get_string_value` are shown in Table B.153.

Table B.153 – Builtin `get_string_value`

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	m	The ID of a VARIABLE reference.
member_id	unsigned long	I	m	The member ID of a BLOCK_A, PARAMETER_LIST, RECORD, VALUE_ARRAY, COLLECTION or REFERENCE_ARRAY.
string	char*	O	m	The buffer of result string.
len	long	O	m	The size of the result string.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.154 Builtin `get_string_value2`

Purpose

The Builtin `get_string_value2` retrieves the current scaled value of the specified VARIABLE from the VARIABLE table (caching EDD application) or device and returns it.

The parameters `id` and `member_id` are specified using the Builtins `ITEM_ID` and `MEMBER_ID` and the name or reference of the VARIABLE in question.

Lexical structure

```
get_string_value2(blk_id, blk_num, id, member_id, string, len)
```

The lexical elements of the Builtin `get_string_value2` are shown in Table B.154.

Table B.154 – Builtin `get_string_value2`

Parameter name	Type	Direction	Usage	Description
blk_id	unsigned long	I	m	The item ID of the BLOCK_A instance to access.
blk_num	unsigned long	I	m	The occurrence number of the BLOCK_A instance to access.
id	unsigned long	I	m	The ID of a VARIABLE reference.
member_id	unsigned long	I	m	The member ID of a BLOCK_A, PARAMETER_LIST, RECORD, VALUE_ARRAY, COLLECTION or REFERENCE_ARRAY.
string	char*	O	m	specifies the buffer of result string.
len	long	O	m	The size of the result string.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.155 Builtin `GET_TICK_COUNT`

Purpose

The Builtin `GET_TICK_COUNT` returns the time in ms since the last system boot. It can be used for time stamps.

Lexical structure

```
GET_TICK_COUNT(VOID)
```

The lexical element of the Builtin `GET_TICK_COUNT` is shown in Table B.155.

Table B.155 – Builtin GET_TICK_COUNT

Parameter name	Type	Direction	Usage	Description
<return>	long	O	m	The time in ms since the last system boot.

B.156 Builtin get_unsigned**Purpose**

The Builtin get_unsigned retrieves the current unscaled unsigned value of the VARIABLE being operated on.

The Builtin get_unsigned can be called only by methods associated with a pre-edit, post-edit, post-read, or pre-write method. The method associated with pre-edit, post-edit, post-read, or pre-write actions are run when a VARIABLE, which has one of these actions defined as an attribute, is read by an application, a user method, or an edit method.

Gets the current value of an unsigned variable as if it was directly read from a device.

Lexical structure

get_unsigned(value)

The lexical elements of the Builtin get_unsigned are shown in Table B.156.

Table B.156 – Builtin get_unsigned

Parameter name	Type	Direction	Usage	Description
value	unsigned long	O	m	The returned value.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.157 Builtin get_unsigned_lelem**Purpose**

The Builtin get_unsigned_lelem extracts an unsigned value from a LIST.

Lexical structure

get_unsigned_lelem(list_id, index, element_id, subelement_id)

The lexical element of the Builtin get_unsigned_lelem is shown in Table B.157.

Table B.157 – Builtin get_unsigned_lelem

Parameter name	Type	Direction	Usage	Description
list_id	unsigned long	I	m	The item id of the list.
index	int	I	m	The index of the list containing the item to be extracted.
element_id	unsigned long	I	m	The item id of the element to be extracted.
subelement_id	unsigned long	I	m	The member id of the element to be extracted.
<return>	unsigned long	O	m	The specified value.

B.158 Builtin get_unsigned_lelem2

Purpose

The Builtin get_unsigned_lelem2 extracts an unsigned value from an embedded list.

Lexical structure

```
get_unsigned_lelem2(list_id, index, embedded_list_id, embedded_list_index,
element_id, subelement_id)
```

The lexical element of the Builtin get_unsigned_lelem2 is shown in Table B.158.

Table B.158 – Builtin get_unsigned_lelem2

Parameter name	Type	Direction	Usage	Description
list_id	unsigned long	l	m	The item id of the list.
index	int	l	m	The index of the list containing the item to be extracted.
embedded_list_id	unsigned long	l	m	The item id of the embedded list.
embedded_list_index	int	l	m	The index of the embedded list containing the item to be extracted.
element_id	unsigned long	l	m	The item id of the element to be extracted.
subelement_id	unsigned long	l	m	The member id of the element to be extracted.
<return>	unsigned long	O	m	returns the specified value.

B.159 Builtin get_unsigned_value

Purpose

The Builtin get_unsigned_value retrieves the current scaled value of the specified VARIABLE from the VARIABLE table (caching EDD application) or device and returns it.

The parameters id and member-id are specified using the Builtins ITEM_ID and MEMBER_ID and the name or reference of the variable in question. The variable shall have a data type of unsigned. An unsigned value may contain device variables of the following data types:

- UNSIGNED
- ENUMERATED
- BIT_ENUMERATED
- INDEX

Device variables with data types of enumerated, bit_enumerated, and index shall be handled by the method because they are not supported by ANSI-C. The variable shall be a valid parameter.

Lexical structure

```
get_unsigned_value(id, member_id, value)
```

The lexical elements of the Builtin get_unsigned_value are shown in Table B.159.

Table B.159 – Builtin get_unsigned_value

Parameter name	Type	Direction	Usage	Description
id	unsigned long	l	m	The item ID of the variable to access.
member_id	unsigned	l	m	The member ID of the variable to access.

Parameter name	Type	Direction	Usage	Description
	long			
value	unsigned long	O	m	The returned value of the variable.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.160 Builtin get_unsigned_value2

Purpose

The Builtin `get_unsigned_value2` retrieves the current scaled value of the specified VARIABLE from the VARIABLE table (caching EDD application) or device and returns it.

The parameters `id` and `member-id` are specified using the Builtins `ITEM_ID` and `MEMBER_ID` and the name or reference of the variable in question. The variable shall have a data type of unsigned. An unsigned value may contain device variables of the following data types:

- UNSIGNED
- ENUMERATED
- BIT_ENUMERATED
- INDEX

Device variables with data types of enumerated, bit enumerated, and index shall be handled by the method because they are not supported by ANSI-C. The variable shall be a valid parameter.

Lexical structure

```
get_unsigned_value2(blk_id, blk_num, id, member_id, value)
```

The lexical elements of the Builtin `get_unsigned_value2` are shown in Table B.160.

Table B.160 – Builtin `get_unsigned_value2`

Parameter name	Type	Direction	Usage	Description
blk_id	unsigned long	I	m	The item ID of the BLOCK_A instance to access.
blk_num	unsigned long	I	m	The occurrence number of the BLOCK_A instance to access.
id	unsigned long	I	m	The item ID of the variable to access.
member_id	unsigned long	I	m	The member ID of the variable to access.
value	unsigned long	O	m	The returned value of the variable.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.161 Builtin get_variable_string

Purpose

The Builtin `get_variable_string` will get a textual representation of the referenced VARIABLE.

Lexical structure

```
get_variable_string(device_var_name, destination_string)
```

The lexical elements of the Builtin `get_variable_string` are shown in Table B.161.

Table B.161 – Builtin get_variable_string

Parameter name	Type	Direction	Usage	Description
device_var_name	reference	I	m	The name of the referenced VARIABLE.
destination_string	string	O	m	Buffer for the textual representation of the referenced VARIABLE.
<return>	int	O	m	One of the return codes specified in Table B.347.

B.162 Builtin GetCurrentDate**Purpose**

The builtin GetCurrentDate returns the EDD application's best approximation of the current calendar date.

Lexical Structure

GetCurrentDate()

The lexical elements of the Builtin GetCurrentDate are shown in Table B.162.

Table B.162 – Builtin GetCurrentDate

Parameter name	Type	Direction	Usage	Description
<return>	time_t	O	m	The current calendar date.

B.163 Builtin GetCurrentDateAndTime**Purpose**

The builtin GetCurrentDateAndTime returns the EDD application's best approximation of the current calendar date and time.

Lexical Structure

GetCurrentDateAndTime()

The lexical elements of the Builtin GetCurrentDateAndTime are shown in Table B.163.

Table B.163 – Builtin GetCurrentDateAndTime

Parameter name	Type	Direction	Usage	Description
<return>	time_t	O	m	The current calendar date and time.

B.164 Builtin GetCurrentTime**Purpose**

The Builtin GetCurrentTime gets the current time from the host application's best approximation of the current calendar time.

Lexical Structure

GetCurrentTime(VOID)

The lexical elements of the Builtin GetCurrentTime are shown in Table B.164.

Table B.164 – Builtin GetCurrentTime

Parameter name	Type	Direction	Usage	Description
<return>	time_t	O	m	The current calendar time.

B.165 Builtin iassign

Purpose

The Builtin iassign will assign the specified value to the device variable. The variable shall be valid, and shall reference a variable of type int.

Lexical structure

iassign(device_var, new_value)

The lexical elements of the Builtin iassign are shown in Table B.165.

Table B.165 – Builtin iassign

Parameter name	Type	Direction	Usage	Description
device_var	reference	I	m	The name of the variable to access.
new_value	int	I	m	The new value of the VARIABLE.
<return>	int	O	m	One of the return codes specified in Table B.347.

B.166 Builtin igetval

Purpose

The Builtin igetval is specifically designed to be used in pre-read/write and post-read/write methods. This function will return the value of a device variable as it was received from the connected device. Scaling operations may then be performed on the variable value before it is stored in the EDD application.

Lexical structure

igetval(VOID)

The lexical element of the Builtin igetval is shown in Table B.166.

Table B.166 – Builtin igetval

Parameter name	Type	Direction	Usage	Description
<return>	int	O	m	The value of the variable.

B.167 Builtin IGNORE_ALL_COMM_STATUS

Purpose

The Builtin IGNORE_ALL_COMM_STATUS will clear all of the bits in the comm status retry and abort masks. This will cause the system to ignore all bits in the comm status value. Comm status is defined to be the first data octet returned in a transaction, when bit 7 of this octet is set.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the Builtin send_command for implementation of the masks. See ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this function is used by the Builtins send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status and display.

Lexical structure

IGNORE_ALL_COMM_STATUS(VOID)

The lexical element of the Builtin IGNORE_ALL_COMM_STATUS is shown in Table B.167.

Table B.167 – Builtin IGNORE_ALL_COMM_STATUS

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.168 Builtin IGNORE_ALL_DEVICE_STATUS**Purpose**

The Builtin IGNORE_ALL_DEVICE_STATUS will clear all of the bits in the device status retry and abort masks. This will cause the system to ignore all bits in the device status octet. Device status is defined to be the second data octet returned in a transaction.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the Builtin send_command for implementation of the masks. See ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this function is used by the Builtins send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status and display.

Lexical structure

IGNORE_ALL_DEVICE_STATUS (VOID)

The lexical element of the Builtin IGNORE_ALL_DEVICE_STATUS is shown in Table B.168.

Table B.168 – Builtin IGNORE_ALL_DEVICE_STATUS

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.169 Builtin IGNORE_ALL_RESPONSE_CODES**Purpose**

The Builtin IGNORE_ALL_RESPONSE_CODES will clear all of the bits in the response code retry and abort masks. This will cause the system to ignore all response code values returned from the device.

The response code is defined to be the first data octet returned in a transaction, when bit 7 of this octet is 0.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the Builtin send_command for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this function is used by the Builtins send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status and display.

Lexical structure

IGNORE_ALL_RESPONSE_CODES (VOID)

The lexical element of the Builtin IGNORE_ALL_RESPONSE_CODES is shown in Table B.169.

Table B.169 – Builtin IGNORE_ALL_RESPONSE_CODES

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.170 Builtin IGNORE_COMM_ERROR

Purpose

The Builtin IGNORE_COMM_ERROR will set the comm error mask such that a communications error condition will be ignored while sending a command.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this function is used by the Builtins send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status and display.

Lexical structure

IGNORE_COMM_ERROR(VOID)

The lexical element of the Builtin IGNORE_COMM_ERROR is shown in Table B.170.

Table B.170 – Builtin IGNORE_COMM_ERROR

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.171 Builtin IGNORE_COMM_STATUS

Purpose

The Builtin IGNORE_COMM_STATUS will clear the correct bit(s) in the comm status abort and retry mask such that the specified bits in the comm_status value will be ignored. comm_status is defined to be the first data octet returned in a transaction, when bit 7 of this octet is 1.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this function is used by the Builtins send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status and display.

Lexical structure

IGNORE_COMM_STATUS(comm_status)

The lexical elements of the Builtin IGNORE_COMM_STATUS are shown in Table B.171.

Table B.171 – Builtin IGNORE_COMM_STATUS

Parameter name	Type	Direction	Usage	Description
comm_status	int	l	m	The new retry and abort mask for ignoring.
<return>	VOID	—	—	—

B.172 Builtin IGNORE_DEVICE_STATUS

Purpose

The Builtin IGNORE_DEVICE_STATUS will clear the correct bit(s) in the device status abort and retry masks such that the specified bits in the device status octet are ignored. Device status is defined to be the second data octet returned in a transaction.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this function is used by the Builtins send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status and display.

Lexical structure

IGNORE_DEVICE_STATUS(device_status)

The lexical elements of the Builtin IGNORE_DEVICE_STATUS are shown in Table B.172.

Table B.172 – Builtin IGNORE_DEVICE_STATUS

Parameter name	Type	Direction	Usage	Description
device_status	int	l	m	The new retry and abort mask for ignoring.
<return>	VOID	—	—	—

B.173 Builtin IGNORE_NO_DEVICE

Purpose

The Builtin IGNORE_NO_DEVICE will set the no device mask to show that the no-device condition should be ignored while sending a command.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this function is used by the Builtins send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status and display.

Lexical structure

IGNORE_NO_DEVICE(VOID)

The lexical element of the Builtin IGNORE_NO_DEVICE is shown in Table B.173.

Table B.173 – Builtin IGNORE_NO_DEVICE

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.174 Builtin IGNORE_RESPONSE_CODE

Purpose

The Builtin IGNORE_RESPONSE_CODE will clear the correct bit(s) in the response code masks such that the specified response code value will be ignored. The response code is defined to be the first data octet returned in a transaction, when bit 7 of this octet is 0.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this function is used by the Builtins send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status and display.

Lexical structure

IGNORE_RESPONSE_CODE(response_code)

The lexical elements of the Builtin IGNORE_RESPONSE_CODE are shown in Table B.174.

Table B.174 – Builtin IGNORE_RESPONSE_CODE

Parameter name	Type	Direction	Usage	Description
response_code	int	I	m	specifies the new retry and abort mask for ignoring
<return>	VOID	—	—	—

B.175 Builtin int_value

Purpose

The Builtin int_value will return the value of the specified device variable. The variable identifier shall be valid and of type integer.

Lexical structure

int_value(source_var_name)

The lexical elements of the Builtin int_value are shown in Table B.175.

Table B.175 – Builtin int_value

Parameter name	Type	Direction	Usage	Description
source_var_name	char[]	I	m	The name of the VARIABLE.
<return>	int	O	m	The value of the variable.

B.176 Builtin is_NaN

Purpose

The Builtin is_NaN checks whether a double floating-point value is not a number. Not a number is specified by the IEEE 754 specification as an invalid double value. The value for NaN for a double value is: 0x7FF8000000000000

Lexical structure

is_NaN(dvalue)

The lexical elements of the Builtin is_NaN are shown in Table B.176.

Table B.176 – Builtin is_NaN

Parameter name	Type	Direction	Usage	Description
dvalue	double	I	m	The name of the variable.
<return>	long	O	m	One if the dvalue is equal to NaN, zero otherwise.

B.177 Builtin isetval

Purpose

The Builtin isetval will set the value of the device variable of type float for which a pre/post action has been designated to the specified value. isetval is specifically designed to be used in pre-read/write and post-read/write methods. This function will set the value of a device variable to the specified value.

Lexical structure

isetval (value)

The lexical elements of the Builtin isetval are shown in Table B.177.

Table B.177 – Builtin isetval

Parameter name	Type	Direction	Usage	Description
value	int	I	m	The new value of the variable.
<return>	int	O	m	The value of the variable.

B.178 Builtin ITEM_ID

Purpose

The Builtin ITEM_ID can be used in methods to specify the item ID of a variable for calls to other.

Lexical structure

ITEM_ID (name)

The lexical elements of the Builtin ITEM_ID are shown in Table B.178.

Table B.178 – Builtin ITEM_ID

Parameter name	Type	Direction	Usage	Description
name	reference	I	m	The name of the variable.
<return>	unsigned long	O	m	The VARIABLE's item ID.

B.179 Builtin ITOA

Purpose

The Builtin ITOA converts an integer value into a string.

Lexical structure

ITOA (i)

The lexical elements of the Builtin ITOA are shown in Table B.179.

Table B.179 – Builtin ITOA

Parameter name	Type	Direction	Usage	Description
i	int	I	m	An integer value to convert to a string.
<return>	DD_STRING	O	m	The value converted to a string.

B.180 Builtin itoa (version A)**Purpose**

The Builtin converts an integer value into a text string.

Lexical structure

```
itoa(i)
```

The lexical elements of the Builtin itoa are shown in Table B.180.

Table B.180 – Builtin itoa (version A)

Parameter name	Type	Direction	Usage	Description
i	int	I	m	An integer value to convert to text.
<return>	char[]	O	m	The returned text.

B.181 Builtin itoa (version B)**Purpose**

The Builtin itoa converts an integer value into a text string.

Lexical structure

```
itoa(i, destination, radix)
```

The lexical elements of the Builtin itoa are shown in Table B.181.

Table B.181 – Builtin itoa (version B)

Parameter name	Type	Direction	Usage	Description
i	int	I	m	An integer value to convert to text.
destination	char[]	I	m	The string in which text is to be stored.
radix	int	I	m	The base of the value (2 = binary, 8 = octal, 10 = decimal, 16 = hexadecimal).
<return>	char[]	O	m	The returned text.

B.182 Builtin ivar_value**Purpose**

The Builtin ivar_value will return the value of the specified variable. The variable source_var_name shall be valid and of type integer.

Lexical structure

```
ivar_value(source_var_name)
```

The lexical elements of the Builtin ivar_value are shown in Table B.182.

Table B.182 – Builtin ivar_value

Parameter name	Type	Direction	Usage	Description
source_var_name	reference	I	m	The name of the variable.
<return>	int	O	m	The value of the variable.

B.183 Builtin lassign

Purpose

The Builtin lassign will assign the specified value to the device variable. The variable shall be valid, and shall reference a variable of type long.

Lexical structure

```
lassign(var_name, new_value)
```

The lexical elements of the Builtin lassign are shown in Table B.183.

Table B.183 – Builtin lassign

Parameter name	Type	Direction	Usage	Description
var_name	reference	I	m	The name of the VARIABLE.
new_value	int	I	m	The new value of the variable.
<return>	int	O	m	One of the return codes specified in Table B.347.

B.184 Builtin lgetval

Purpose

The Builtin lgetval is specifically designed to be used in pre-read/write and post-read/write methods. This function returns the value of a VARIABLE as it was received from the connected device. Scaling operations may then be performed on the VARIABLE value before it is stored in the EDD application.

Lexical structure

```
lgetval(VIOD)
```

The lexical elements of the Builtin lgetval are shown in Table B.184.

Table B.184 – Builtin lgetval

Parameter name	Type	Direction	Usage	Description
<return>	int	O	m	The value of the variable.

B.185 Builtin ListDeleteElementAt

Purpose

The Builtin ListDeleteElementAt will delete an element from a list. If the list is a member of a FILE, the specified element will be removed from the persistent store.

If a COUNT are explicitly declared on the specified LIST, any variables referenced by the COUNT attribute will not be updated. The method is responsible for updating the COUNT when elements are deleted from the list. If a COUNT are not explicitly declared, the COUNT attribute will be automatically updated.

If the index parameter is less than zero or greater than the number of elements in the list, an error will be returned. If the list parameter does not refer to a LIST instance, an error will be returned.

Lexical structure

```
ListDeleteElementAt(list, index)
```

The lexical element of the Builtin ListDeleteElementAt is shown in Table B.185.

Table B.185 – Builtin ListDeleteElementAt

Parameter name	Type	Direction	Usage	Description
list	reference	I	m	The name of a LIST instance.
index	int	I	m	The index of the element to be deleted.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.186 Builtin ListDeleteElementAt2**Purpose**

The Builtin ListDeleteElementAt2 deletes elements from an embedded list. If the list is a member of a FILE, the specified element will be removed from the persistent store.

Lexical structure

ListDeleteElementAt2(list_id, list_index, embedded_list_id, embedded_index, embedded_count)

The lexical element of the Builtin ListDeleteElementAt2 is shown in Table B.186.

Table B.186 – Builtin ListDeleteElementAt2

Parameter name	Type	Direction	Usage	Description
list_id	unsigned long	I	m	The item of the list.
list_index	int	I	m	The index of the list containing the embedded list.
embedded_list_id	unsigned long	I	m	The item id of the embedded_list.
embedded_index	int	I	m	The index of the first element to delete.
embedded_count	int	I	m	The number of elements to delete.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.187 Builtin ListInsert**Purpose**

The Builtin ListInsert will insert an element to a list. If the list is a member of a FILE, the specified element will be added to the persistent store.

If COUNT and/or CAPACITY attributes are explicitly declared on the specified LIST, any variables referenced by these attributes will not be updated. The method is responsible for updating the COUNT and CAPACITY when elements are inserted in the list. If a COUNT and/or CAPACITY attributes are not explicitly declared, these attributes will be updated automatically.

If the index parameter is less than zero or greater than the capacity of the list, an error will be returned. If the list parameter does not refer to a LIST instance, an error will be returned.

Lexical structure

ListInsert(list, index, item)

The lexical element of the Builtin ListInsert is shown in Table 150.

Table B.187 – Builtin ListInsert

Parameter name	Type	Direction	Usage	Description
list	reference	I	m	The name of a LIST instance.
index	int	I	m	The index of the requested element.

Parameter name	Type	Direction	Usage	Description
item	reference	I	m	The name of the item to be inserted.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.188 Builtin ListInsert2

Purpose

The Builtin ListInsert2 inserts an element into an embedded list. If the list is a member of a FILE, the specified element will be added to the persistent store.

Lexical structure

ListInsert2(list_id, list_index, embedded_list_id, embedded_index, item_id)

The lexical element of the Builtin ListInsert2 is shown in Table B.188.

Table B.188 – Builtin ListInsert2

Parameter name	Type	Direction	Usage	Description
list_id	unsigned long	I	m	The item of the list.
list_index	int	I	m	The index of the list containing the embedded list.
embedded_list_id	unsigned long	I	m	The item id of the embedded list.
embedded_index	int	I	m	The index where the item should be inserted.
item_id	unsigned long	I	m	The item id of the item to be inserted.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.189 Builtin log

Purpose

The Builtin log returns the natural logarithm of a floating point value.

Lexical structure

log(x)

The lexical elements of the Builtin log are shown in Table B.189.

Table B.189 – Builtin log

Parameter name	Type	Direction	Usage	Description
x	double	I	m	A floating point value.
<return>	double	O	m	The natural logarithm.

B.190 Builtin LOG_MESSAGE

Purpose

The Builtin LOG_MESSAGE inserts a message into a logging protocol.

Lexical structure

LOG_MESSAGE(priority, message)

The lexical elements of the Builtin LOG_MESSAGE are shown in Table B.190.

Table B.190 – Builtin LOG_MESSAGE

Parameter name	Type	Direction	Usage	Description
priority	int	I	m	The priority of the message (0 = error, 1 = warning, 2 = low).
message	char[]	I	m	The message string to insert into the logging protocol.
<return>	VOID	—	—	—

B.191 Builtin log10

Purpose

The Builtin exp returns the base 10 logarithm of a floating point value.

Lexical structure

log10(x)

The lexical elements of the Builtin log10 are shown in Table B.191.

Table B.191 – Builtin log10

Parameter name	Type	Direction	Usage	Description
x	double	I	m	A floating point value.
<return>	double	O	m	The base 10 logarithm.

B.192 Builtin log2

Purpose

The Builtin log2 returns the base 2 logarithm of a floating-point value.

Lexical structure

log2(x)

The lexical elements of the Builtin log2 are shown in Table B.192.

Table B.192 – Builtin log2

Parameter name	Type	Direction	Usage	Description
x	double	I	m	A floating point value.
<return>	double	O	m	The base 2 logarithm.

B.193 Builtin long_value

Purpose

The Builtin long_value will return the value of the specified device variable. The variable identifier shall be valid and of long type.

Lexical structure

long_value(source_var_name)

The lexical elements of the Builtin long_value are shown in Table B.193.

Table B.193 – Builtin long_value

Parameter name	Type	Direction	Usage	Description
source_var_name	reference	I	m	The name of the variable.
<return>	int	O	m	The value of the variable.

B.194 Builtin LongToByte**Purpose**

The Builtin LongToByte will provide a vector of unsigned char values representing the particular elements of a long value.

Lexical structure

```
LongToByte(in, out1, out2, out3, out4)
```

The lexical element of the Builtin LongToByte is shown in Table B.194.

Table B.194 – Builtin LongToByte

Parameter name	Type	Direction	Usage	Description
in	long	I	m	The long value to convert.
out1	unsigned char	O	m	First element of the long value (most significant byte).
out2	unsigned char	O	m	Second element of the long value.
out3	unsigned char	O	m	Third element of the long value.
out4	unsigned char	O	m	Fourth element of the long value (least significant byte).
<return>	VOID	—	—	—

B.195 Builtin lsetval**Purpose**

The Builtin lsetval will set the value of the device variable of long type for which a pre/post action has been designated to the specified value. It is specifically designed to be used in pre-read/write and post-read/write methods. This function will set the value of a device variable to the specified value.

Lexical structure

```
lsetval(value)
```

The lexical elements of the Builtin lsetval are shown in Table B.195.

Table B.195 – Builtin lsetval

Parameter name	Type	Direction	Usage	Description
value	int	I	m	The new value of the VARIABLE.
<return>	int	O	m	The value of the variable.

B.196 Builtin lvar_value**Purpose**

The Builtin lvar_value will return the value of the specified device variable. The variable identifier shall be valid and of long type.

Lexical structure

`lvar_value(source_var_name)`

The lexical elements of the Builtin `lvar_value` are shown in Table B.196.

Table B.196 – Builtin `lvar_value`

Parameter name	Type	Direction	Usage	Description
<code>source_var_name</code>	reference	I	m	The name of the VARIABLE.
<return>	int	O	m	The value of the variable.

B.197 Builtin `Make_Time`

Purpose

The Builtin `Make_Time` creates a `time_t` from its input parameters.

Lexical structure

`Make_Time(year, month, day, hour, minute, second, isDST)`

The lexical element of the Builtin `Make_Time` is shown in Table B.197.

Table B.197 – Builtin `Make_Time`

Parameter name	Type	Direction	Usage	Description
<code>year</code>	int	I	m	The year.
<code>month</code>	int	I	m	The month.
<code>day</code>	int	I	m	The day.
<code>hour</code>	int	I	m	The hour.
<code>minute</code>	int	I	m	The minute.
<code>second</code>	int	I	m	The second.
<code>isDST</code>	int	I	m	Specifies whether daylight savings time is in effect. If zero, daylight savings time is not in effect; otherwise, daylight savings time is in effect.
<return>	<code>time_t</code>	O	m	The <code>time_t</code> that was created or -1 is an invalid parameter is detected.

B.198 Builtin `MEMBER_ID`

Purpose

The Builtin `MEMBER_ID` can be used in methods to specify the member ID of a variable for calls to other Builtins. It returns the member ID of a BLOCK, PARAMETER_LIST, RECORD, REFERENCE_ARRAY, COLLECTION or VALUE_ARRAY.

Lexical structure

`MEMBER_ID(name)`

The lexical elements of the Builtin `MEMBER_ID` are shown in Table B.198.

Table B.198 – Builtin `MEMBER_ID`

Parameter name	Type	Direction	Usage	Description
<code>name</code>	reference	I	m	The name of an item member.
<return>	unsigned long	O	m	The referenced member ID.

B.199 Builtin MenuDisplay

Purpose

The Builtin MenuDisplay will display a MENU instance to the user and a supplied list of options for the user to select from. Once a selection is made, the option position in the option list is returned. The position in the list is zero-based. For example, if the provided list contains two items and the first item is selected, this function will return a 0. If the second item is selected, a 1 is returned. The options are passed to the function as a single string with semicolon separators between the options.

Lexical structure

MenuDisplay(menu, options, selection)

If the EDD application is unable to display the menu, an error will be returned.

The lexical element of the Builtin MenuDisplay is shown in Table B.199.

Table B.199 – Builtin MenuDisplay

Parameter name	Type	Direction	Usage	Description
menu	reference	I	m	The MENU to be displayed.
options	char[]	I	m	The option string, consisting of separated text, that defines a set of options presented to the user.
selection	long*	O	m	The zero-based selected option (see options parameter).
<return>	long	O	m	One of the return codes specified in Table B.346.

B.200 Builtin method_abort

Purpose

The Builtin method_abort displays a message indicating that the method is aborting and waits for acknowledgment from the user.

After the acknowledgment, the abort methods in the abort method list are executed one after another from the front of the list (where the first abort method was added to the list) to the back of the list. The abort method list is maintained only for the execution of a single invocation of the method.

After the abort methods are executed, the method terminates and control is returned to the application.

If an abort method calls method_abort, the abort method list processing is immediately terminated.

Lexical structure

method_abort(prompt)

The lexical elements of the Builtin method_abort are shown in Table B.200.

Table B.200 – Builtin method_abort

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	A message to be displayed.
<return>	VOID	—	—	—

B.201 Builtin nan

Purpose

The Builtin nan returns a "not a number" value as a double with a fractional bit pattern based on a string. If the character sequence begins with a quiet or signaling NaN prefix letter, then any following hex digits in the character sequence must be valid for that type of NaN.

The string can be:

- "" (an empty string) generates the default NaN 0x7FA00000.
- "Q" or "q" equals the default quiet NaN.
- "S" or "s" equals the default signaling NaN.
- "I_XXXXXX" equals where I is either a "Q", "q", "S", or "s" followed by an underscore and the most significant hex digits of the fraction. Trailing zeros may be omitted.

Lexical structure

nan(value)

The lexical element of the Builtin nan is shown in Table B.201.

Table B.201 – Builtin nan

Parameter name	Type	Direction	Usage	Description
value	DD_STRING	I	m	The "not a number" value to be returned.
<return>	double	O	m	The specified value.

B.202 Builtin NaN_value

Purpose

The Builtin NaN_value returns the value of "not a number".

Lexical structure

NaN_value(value)

The lexical element of the Builtin NaN_value is shown in Table B.202.

Table B.202 – Builtin NaN_value

Parameter name	Type	Direction	Usage	Description
value	double	O	m	A pointer through which the value is returned.
<return>	long	O	m	The specified value.

B.203 Builtin nanf

Purpose

The Builtin nanf returns a "not a number" value as a float with a fractional bit pattern based on a string. If the character sequence begins with a quiet or signaling NaN prefix letter, then any following hex digits in the character sequence must be valid for that type of NaN.

The string can be:

- "" (an empty string), generates the default NaN 0x7FA00000.
- "Q" or "q" equals the default quiet NaN.
- "S" or "s" equals the default signaling NaN.

- "I_xxxxxx" equals where I is either a "Q", "q", "S", or "s" followed by an underscore and the most significant hex digits of the fraction. Trailing zeros may be omitted.

Lexical structure

`nanf(value)`

The lexical element of the Builtin `nanf` is shown in Table B.203.

Table B.203 – Builtin `nanf`

Parameter name	Type	Direction	Usage	Description
value	DD_STRIN G	I	m	The "not a number" value to be returned.
<return>	float	O	m	The specified value.

B.204 Builtin `ObjectReference`

Purpose

The Builtin `ObjectReference` returns unique id of an object. With the return object reference it is possible to have access to the variable and methods of other objects that may have different EDD.

NOTE For example, in an EDD of a remote I/O head station, it is possible to make a consistency check with the modules using `ObjectReference(CHILD)`.

Lexical structure

`ObjectReference(string)`

The lexical element of the Builtin `ObjectReference` is shown in Table B.204.

Table B.204 – Builtin `process_abort`

Parameter name	Type	Direction	Usage	Description
string	char[]	I	m	For the direction of navigation or the name of an object, see Table 173.
<return>	long	O	m	An ID which represent an object reference.

B.205 Builtin `pop_abort_method`

Purpose

The Builtin `pop_abort_method` removes an abort method from the top of the abort method list, which is the list of methods to be executed if the current method is aborted.

Lexical Structure

`pop_abort_method(VOID)`

The lexical element of the Builtin `pop_abort_method` is shown in Table B.205.

Table B.205 – Builtin `pop_abort_method`

Parameter name	Type	Direction	Usage	Description
<return>	int	o	m	One of the return codes specified in Table B.347.

B.206 Builtin `pow`

Purpose

The Builtin `pow` returns the power function of two floating-point values.

Lexical structure

`pow(x, y)`

The lexical elements of the Builtin `pow` are shown in Table B.206.

Table B.206 – Builtin `pow`

Parameter name	Type	Direction	Usage	Description
<code>x</code>	double	I	m	The base.
<code>y</code>	double	I	m	The exponent.
<return>	double	O	m	The value of <code>x</code> raised to the <code>y</code> power.

B.207 Builtin `process_abort`

Purpose

The Builtin `process_abort` will abort the current method, running any abort methods, which are in the abort method list. Unlike the Builtin `abort`, no message will be displayed when this function is executed. This Builtin function may not be run from inside an abort method.

Lexical structure

`process_abort(VOID)`

The lexical element of the Builtin `process_abort` is shown in Table B.207.

Table B.207 – Builtin `process_abort`

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.208 Builtin `push_abort_method`

Purpose

The Builtin `push_abort_method` adds an abort method to the top of the abort method list, which is the list of methods to be executed if the current method is aborted.

Lexical Structure

`push_abort_method(abort_method_id)`

The lexical element of the Builtin `push_abort_method` is shown in Table B.208.

Table B.208 – Builtin `push_abort_method`

Parameter name	Type	Direction	Usage	Description
<code>abort_method_id</code>	unsigned long	I	m	The abort method to be added.
<return>	int	O	m	One of the return codes specified in Table B.347.

B.209 Builtin `put_date`

Purpose

The Builtin `put_date` stores the new un-scaled date value of the variable being processed.

The Builtin `put_date` can be called only by methods associated with a pre-edit, post-edit, post-read, or pre-write action. The methods associated with pre-edit, post-edit, post-read, or pre-write actions are run when a variable, which has one of these actions defined as an attribute, is written by an application, a user method, or an edit method.

Lexical structure

```
put_date(data, size)
```

The lexical elements of the Builtin `put_date` are shown in Table B.209.

Table B.209 – Builtin `put_date`

Parameter name	Type	Direction	Usage	Description
data	char[]	I	m	The new value of the VARIABLE.
size	long	I	m	The length of the new value.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.210 Builtin `put_date_value`

Purpose

The Builtin `put_date_value` stores the provided value of the specified scaled variable into the variable table (caching EDD application) or device.

The parameters `id` and `member id` are specified using the Builtins `ITEM_ID` and `MEMBER_ID` and the name or reference of the variable in question.

It puts the specified value of the `date_and_time`, `time`, or `duration` variable into the variable table or device.

The variable shall have a data type of `char`. The `char` string may contain up to eight chars and contain a variable of the following data types:

- `DATA_AND_TIME`
- `TIME`
- `DURATION`

The variable shall be valid in the variable table or device.

Lexical structure

```
put_date_value(id, member_id, data, size)
```

The lexical elements of the Builtin `put_date_value` are shown in Table B.210.

Table B.210 – Builtin `put_date_value`

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	o	The item ID of the VARIABLE to access.
member_id	unsigned long	I	o	The member ID of the VARIABLE access.
data	char[]	I	m	The new value of the VARIABLE.
size	long	I	m	The length of the new value.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.211 Builtin `put_date_value2`

Purpose

The Builtin `put_date_value2` stores the provided value of the specified scaled variable into the variable table (caching EDD application) or device.

The parameters `id` and `member id` are specified using the Builtins `ITEM_ID` and `MEMBER_ID` and the name or reference of the variable in question.

It puts the specified value of the date_and_time, time, or duration variable into the variable table or device.

The variable shall have a data type of char. The char string may contain up to eight chars and contain a variable of the following data types:

- DATA_AND_TIME
- TIME
- DURATION

The variable shall be valid in the variable table or device.

Lexical structure

put_date_value2(blk_id, blk_num, id, member_id, data, size)

The lexical elements of the Builtin put_date_value2 are shown in Table B.211.

Table B.211 – Builtin put_date_value2

Parameter name	Type	Direction	Usage	Description
blk_id	unsigned long	I	m	The item ID of the BLOCK_A instance to access.
blk_num	unsigned long	I	m	The occurrence number of the BLOCK_A instance to access.
id	unsigned long	I	o	The item ID of the VARIABLE to access.
member_id	unsigned long	I	o	The member ID of the VARIABLE access.
data	char[]	I	m	The new value of the VARIABLE.
size	long	I	m	The length of the new value.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.212 Builtin put_double

Purpose

The Builtin put_double can be called only by methods associated with a pre-edit, post-edit post-read, or pre-write action.

The methods associated with pre-edit, post-edit, post-read, or pre-write actions are run when a variable, which has one of these actions defined as an attribute, is written by an application, a user method, or an edit method.

Stores the new value of the double variable as it resides in the device (un-scaled). The Builtin put_double stores the new un-scaled double value of the variable being processed.

Lexical structure

put_double(value)

The lexical elements of the Builtin put_double are shown in Table B.212.

Table B.212 – Builtin put_double

Parameter name	Type	Direction	Usage	Description
value	double	I	m	The new value of the VARIABLE.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.213 Builtin put_double_value

Purpose

The Builtin put_double_value stores the provided value of the specified scaled variable into the variable table (caching EDD application) or device.

The parameters id and member_id are specified using the Builtins ITEM_ID and MEMBER_ID and the name or reference of the variable in question.

The specified value of a double variable is put into the variable table or device. The variable shall have a data type of double and be a valid variable in the variable table or device.

Lexical structure

put_double_value(id, member_id, value)

The lexical elements of the Builtin put_double_value are shown in Table B.213.

Table B.213 – Builtin put_double_value

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	o	The item ID of the VARIABLE to access.
member_id	unsigned long	I	o	The member ID of the VARIABLE access.
value	double	I	m	The new value of the VARIABLE.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.214 Builtin put_double_value2

Purpose

The Builtin put_double_value2 stores the provided value of the specified scaled variable into the variable table (caching EDD application) or device.

The parameters id and member_id are specified using the Builtins ITEM_ID and MEMBER_ID and the name or reference of the variable in question. ? Puts the specified value of a double variable into the variable table or device. The variable shall have a data type of double and be a valid variable in the variable table or device.

Lexical structure

put_double_value2(blk_id, blk_num, id, member_id, value)

The lexical elements of the Builtin put_double_value2 are shown in Table B.214.

Table B.214 – Builtin put_double_value2

Parameter name	Type	Direction	Usage	Description
blk_id	unsigned long	I	m	The item ID of the BLOCK_A instance to access.
blk_num	unsigned long	I	m	The occurrence number of the BLOCK_A instance to access.
id	unsigned long	I	o	The item ID of the VARIABLE to access.
member_id	unsigned long	I	o	The member ID of the VARIABLE access.
value	double	I	m	The new value of the VARIABLE.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.215 Builtin put_float

Purpose

The Builtin put_float stores the new unscaled floating-point value of the variable being operated on.

The Builtin put_float can be called only by methods associated with a pre-edit, post-edit, post-read, or pre-write action. The methods associated with pre-edit, post-edit, post-read, or pre-write actions are run when a variable, which has one of these actions defined as an attribute, is written by an application, a user method, or an edit method.

Lexical structure

put_float(value)

The lexical elements of the Builtin put_float are shown in Table B.215.

Table B.215 – Builtin put_float

Parameter name	Type	Direction	Usage	Description
value	double	I	m	The new value of the VARIABLE.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.216 Builtin put_float_value

Purpose

The Builtin put_float_value stores the provided value of the specified scaled variable in the variable table (caching EDD application) or the device.

The type of value is double to accommodate automatic promotion, which could occur whenever two floating-point values are multiplied.

The parameters id and member_id are specified using the Builtins ITEM_ID and MEMBER_ID Builtins and the name or reference of the variable in question. The variable shall have a data type of float and be a valid variable in the variable table or device.

Lexical structure

put_float_value(id, member_id, value)

The lexical elements of the Builtin put_float_value are shown in Table B.216.

Table B.216 – Builtin put_float_value

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	o	The item ID of the VARIABLE to access.
member_id	unsigned long	I	o	The member ID of the VARIABLE access.
value	double	I	m	The new value of the VARIABLE.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.217 Builtin put_float_value2

Purpose

The Builtin put_float_value2 stores the provided value of the specified scaled variable in the variable table (caching EDD application) or the device.

The type of value is double to accommodate automatic promotion, which could occur whenever two floating-point values are multiplied.

The parameters `id` and `member_id` are specified using the Builtins `ITEM_ID` and `MEMBER_ID` Builtins and the name or reference of the variable in question. The variable shall have a data type of float and be a valid variable in the variable table or device.

Lexical structure

```
put_float_value2(blk_id, blk_num, id, member_id, value)
```

The lexical elements of the Builtin `put_float_value2` are shown in Table B.217.

Table B.217 – Builtin `put_float_value2`

Parameter name	Type	Direction	Usage	Description
<code>blk_id</code>	unsigned long	I	m	The item ID of the <code>BLOCK_A</code> instance to access.
<code>blk_num</code>	unsigned long	I	m	The occurrence number of the <code>BLOCK_A</code> instance to access.
<code>id</code>	unsigned long	I	o	The item ID of the <code>VARIABLE</code> to access.
<code>member_id</code>	unsigned long	I	o	The member ID of the <code>VARIABLE</code> access.
<code>value</code>	double	I	m	The new value of the <code>VARIABLE</code> .
<return>	long	O	m	One of the return codes specified in Table B.346.

B.218 Builtin `PUT_MESSAGE`

Purpose

The Builtin `PUT_MESSAGE` will display the specified message on the screen. Any embedded local variable references will be expanded to the variable's value (see Builtin `put_message` for Lexical Structure).

Embedded device variables are NOT supported in this function.

NOTE 1 Previously displayed messages are not guaranteed to remain on the screen.

NOTE 2 Some EDD applications will clear the display upon each call to this function.

Lexical structure

```
PUT_MESSAGE(message)
```

The lexical elements of the Builtin `PUT_MESSAGE` are shown in Table B.218.

Table B.218 – Builtin `PUT_MESSAGE`

Parameter name	Type	Direction	Usage	Description
<code>message</code>	char[]	I	m	A message to be displayed.
<return>	VOID	—	—	—

B.219 Builtin `put_message`

Purpose

The Builtin `put_message` will display the specified message on the screen. Any embedded variable references will be expanded to the current value of the variable.

Any number of identifiers may be accessed by the string, but the array entries accessed shall contain valid variable identifiers. Local and device variables may be mixed in the display message.

Multiple languages are also supported in the `put_message` string. This is done by separating the different language strings by their language code identifier. The language code identifier

is the vertical bar character, '|', followed immediately by that country's three-digit phone code. If the phone code is less than three digits in length, it should be padded on the left with zeros. For example, the following string is defined for both English and German.

NOTE 1 Previously displayed messages are not guaranteed to remain on the screen.

NOTE 2 Some EDD applications will clear the display upon each call to this function.

Lexical structure

```
put_message(message, global_var_ids)
```

The lexical elements of the Builtin put_message are shown in Table B.219.

Table B.219 – Builtin put_message

Parameter name	Type	Direction	Usage	Description
message	char[]	I	m	A message to be displayed.
global_var_ids	int[]	I	m	An array which contains unique identifiers of VARIABLE instance.
<return>	VOID	—	—	—

B.220 Builtin put_signed

Purpose

The Builtin put_signed stores the new un-scaled signed value of the variable being operated on.

The Builtin put_signed can be called only by methods associated with a pre-edit, post-edit, post-read, or pre-write action. The methods associated with pre-edit, post-edit, post-read, or pre-write actions are run when a variable, which has one of these actions defined as an attribute, is written by an application, a user method or an edit method.

Lexical structure

```
put_signed(value)
```

The lexical elements of the Builtin put_signed are shown in Table B.220.

Table B.220 – Builtin put_signed

Parameter name	Type	Direction	Usage	Description
value	long	I	m	The new value of the VARIABLE.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.221 Builtin put_signed_value

Purpose

The Builtin put_signed_value stores the new value of the specified scaled variable into the variable table (caching EDD application) or device.

The parameters id and member_id are specified using the Builtins ITEM_ID and MEMBER_ID and the name or reference of the variable in question. Puts the specified value of a signed integer variable into the variable table or device.

The variable to be written shall have a long data type and be a valid variable in the variable table or device.

Lexical structure

```
put_signed_value(id, member_id, value)
```

The lexical elements of the Builtin put_signed_value are shown in Table B.221.

Table B.221 – Builtin put_signed_value

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	o	The item ID of the VARIABLE to access.
member_id	unsigned long	I	o	The member ID of the VARIABLE access.
value	long	I	m	The new value of the VARIABLE.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.222 Builtin put_signed_value2**Purpose**

The Builtin put_signed_value2 stores the new value of the specified scaled variable into the variable table (caching EDD application) or device.

The parameters id and member_id are specified using the Builtins ITEM_ID and MEMBER_ID and the name or reference of the variable in question. ? Puts the specified value of a signed integer variable into the variable table or device.

The variable to be written shall have a long data type and be a valid variable in the variable table or device.

Lexical structure

```
put_signed_value2(blk_id, blk_num, id, member_id, value)
```

The lexical elements of the Builtin put_signed_value2 are shown in Table B.222.

Table B.222 – Builtin put_signed_value2

Parameter name	Type	Direction	Usage	Description
blk_id	unsigned long	I	m	The item ID of the BLOCK_A instance to access.
blk_num	unsigned long	I	m	The occurrence number of the BLOCK_A instance to access.
id	unsigned long	I	o	The item ID of the VARIABLE to access.
member_id	unsigned long	I	o	The member ID of the VARIABLE access.
value	long	I	m	The new value of the VARIABLE.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.223 Builtin put_string**Purpose**

The Builtin put_string stores the new unscaled string value of the variable being operated on.

The variable shall be valid in the variable table or device.

The variable shall have a char data type; a char string may contain device variables of the following data types:

- ASCII
- PASSWORD
- EUC (Extended Unit Code)
- BITSTRING

Device variables with data types of enumerated, bit_enumerated, and index shall be handled by the method because they are not supported by ANSI-C. The Builtin put_string can be called only by methods associated with a pre-edit, post-edit, post-read, or pre-write action. The methods associated with pre-edit, post-edit, post-read, or pre-write actions are run when a variable, which has one of these actions defined as an attribute, is written by an application, a user method or an edit method.

Lexical structure

```
put_string(string, len)
```

The lexical elements of the Builtin put_string are shown in Table B.223.

Table B.223 – Builtin put_string

Parameter name	Type	Direction	Usage	Description
string	char[]	I	m	The new value of the VARIABLE.
len	long	I	m	The length of the new value.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.224 Builtin put_string_value

Purpose

The Builtin put_string_value stores the provided value of the specified scaled variable into the variable table (caching EDD application) or device.

The parameters id and member_id are specified using the Builtins ITEM_ID and MEMBER_ID and the name or reference of the variable in question.

The variable shall be valid in the variable table or device. The variable shall have a data type of char; a char string may contain device variables of the following data types:

- ASCII
- PASSWORD
- EUC (Extended Unit Code)
- BITSTRING

Device variables with data types of enumerated, bit_enumerated, and index shall be handled by the method because they are not supported by ANSI-C.

Lexical structure

```
put_string_value(id, member_id, string, len)
```

The lexical elements of the Builtin put_string_value are shown in Table B.224.

Table B.224 – Builtin put_string_value

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	o	The item ID of the VARIABLE to access.
member_id	unsigned long	I	o	The member ID of the VARIABLE access.
string	char[]	I	m	The new value of the VARIABLE.
len	long	I	m	The length of the new value.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.225 Builtin put_string_value2

Purpose

The Builtin put_string_value2 stores the provided value of the specified scaled variable into the variable table (caching EDD application) or device.

The parameters id and member_id are specified using the Builtins ITEM_ID and MEMBER_ID and the name or reference of the variable in question.

The variable shall be valid in the variable table or device. The variable shall have a data type of char; a char string may contain device variables of the following data types:

- ASCII
- PASSWORD
- EUC (Extended Unit Code)
- BITSTRING

Device variables with data types of enumerated, bit_enumerated, and index shall be handled by the method because they are not supported by ANSI-C.

Lexical structure

```
put_string_value2(blk_id, blk_num, id, member_id, string, len)
```

The lexical elements of the Builtin put_string_value2 are shown in Table B.225.

Table B.225 – Builtin put_string_value2

Parameter name	Type	Direction	Usage	Description
blk_id	unsigned long	I	m	The item ID of the BLOCK_A instance to access.
blk_num	unsigned long	I	m	The occurrence number of the BLOCK_A instance to access.
id	unsigned long	I	o	The item ID of the VARIABLE to access.
member_id	unsigned long	I	o	The member ID of the VARIABLE access.
string	char[]	I	m	The new value of the VARIABLE.
len	long	I	m	The length of the new value.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.226 Builtin put_unsigned

Purpose

The Builtin put_unsigned stores the new unscaled unsigned value of the variable being operated on. The variable shall have a data type of unsigned and be a valid variable in the variable table or device. An unsigned value may contain device variables of the following data types:

- UNSIGNED
- ENUMERATED
- BIT_ENUMERATED
- INDEX

Device variables of these data types shall be handled by the method because they are not supported by ANSI-C. The Builtin put_unsigned should be called only by methods associated with a pre-edit, post-edit, post-read, or pre-write action. The methods associated with pre-edit,

post-edit, post-read, or pre-write actions are run when a variable, which has one of these actions defined as an attribute, is written by an application or another method.

Lexical structure

```
put_unsigned(value)
```

The lexical elements of the Builtin `put_unsigned` are shown in Table B.226.

Table B.226 – Builtin `put_unsigned`

Parameter name	Type	Direction	Usage	Description
value	unsigned long	I	o	The new value of the VARIABLE.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.227 Builtin `put_unsigned_value`

Purpose

The Builtin `put_unsigned_value` stores the provided value of the specified scaled variable into the variable table (caching EDD application) or device.

The parameters `id` and `member_id` are specified using the Builtins `ITEM_ID` and `MEMBER_ID` and the name or reference of the variable in question.

The variable shall have a data type of unsigned and be a valid variable in the variable table or device. An unsigned value may contain device variables of the following data types:

- UNSIGNED
- ENUMERATED
- BIT_ENUMERATED
- INDEX

Device variables of these data types shall be handled by the method because they are not supported by ANSI-C. The variable shall be valid in the variable table or device.

Lexical structure

```
put_unsigned_value(id, member_id, value)
```

The lexical elements of the Builtin `put_unsigned_value` are shown in Table B.227.

Table B.227 – Builtin `put_unsigned_value`

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	o	The item ID of the VARIABLE to access.
member_id	unsigned long	I	o	The member ID of the VARIABLE access.
value	unsigned long	I	o	The new value of the VARIABLE.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.228 Builtin `put_unsigned_value2`

Purpose

The Builtin `put_unsigned_value2` stores the provided value of the specified scaled variable into the variable table (caching EDD application) or device.

The parameters `id` and `member_id` are specified using the Builtins `ITEM_ID` and `MEMBER_ID` and the name or reference of the variable in question.

The variable shall have a data type of unsigned and be a valid variable in the variable table or device. An unsigned value may contain device variables of the following data types:

- UNSIGNED
- ENUMERATED
- BIT_ENUMERATED
- INDEX

Device variables of these data types shall be handled by the method because they are not supported by ANSI-C. The variable shall be valid in the variable table or device.

Lexical structure

```
put_unsigned_value2(blk_id, blk_num, id, member_id, value)
```

The lexical elements of the Builtin `put_unsigned_value2` are shown in Table B.228.

Table B.228 – Builtin `put_unsigned_value2`

Parameter name	Type	Direction	Usage	Description
<code>blk_id</code>	unsigned long	I	m	The item ID of the BLOCK_A instance to access.
<code>blk_num</code>	unsigned long	I	m	The occurrence number of the BLOCK_A instance to access.
<code>id</code>	unsigned long	I	o	The item ID of the VARIABLE to access.
<code>member_id</code>	unsigned long	I	o	The member ID of the VARIABLE access.
<code>value</code>	unsigned long	I	o	The new value of the VARIABLE.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.229 Builtin `read_value`

Purpose

The Builtin `read_value` causes the value of the parameter to be read from the device.

The Builtin `read_value` does nothing in a non-caching EDD application because a non-caching EDD application does not have a variable table to update.

The Builtin `read_value` subjects to the same read and write timeouts as when an application is reading or writing a value. The parameters `id` and `member_id` are specified using the Builtins `ITEM_ID` and `MEMBER_ID` and the name or reference of the variable in question.

Lexical structure

```
read_value(id, member_id)
```

The lexical elements of the Builtin `read_value` are shown in Table B.229.

Table B.229 – Builtin `read_value`

Parameter name	Type	Direction	Usage	Description
<code>id</code>	unsigned long	I	o	The item ID of the VARIABLE to access.
<code>member_id</code>	unsigned long	I	o	The member ID of the VARIABLE access.

Parameter name	Type	Direction	Usage	Description
<return>	long	O	m	One of the return codes specified in Table B.346.

B.230 Builtin read_value2

Purpose

The Builtin read_value2 causes the parameter's value to be read from the device.

The Builtin read_value does nothing in a non-caching EDD application because a non-caching EDD application does not have a variable table to update.

The Builtin read_value subjects to the same read and write timeouts as when an application is reading or writing a value. The parameters id and member_id are specified using the Builtins ITEM_ID and MEMBER_ID and the name or reference of the variable in question.

Lexical structure

read_value2(blk_id, blk_num, id, member_id)

The lexical elements of the Builtin read_value2 are shown in Table B.230.

Table B.230 – Builtin read_value2

Parameter name	Type	Direction	Usage	Description
blk_id	unsigned long	I	m	The item ID of the BLOCK_A instance to access.
blk_num	unsigned long	I	m	The occurrence number of the BLOCK_A instance to access.
id	unsigned long	I	o	The item ID of the VARIABLE to access.
member_id	unsigned long	I	o	The member ID of the VARIABLE access.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.231 Builtin ReadCommand

Purpose

The Builtin ReadCommand allows reading variables with a given command.

Lexical structure

ReadCommand(command_identifier)

The lexical elements of the Builtin ReadCommand are shown in Table B.231.

Table B.231 – Lexical elements of Builtin ReadCommand

Parameter name	Type	Direction	Usage	Description
command_identifier	reference	I	m	The ID of the command to execute.
<return>	long	O	m	A protocol specific error value, the encoding of which is defined by the specific protocol or device.

B.232 Builtin remove_abort_method (version A)

Purpose

The Builtin remove_abort_method will remove a method from the abort method list, which is the list of methods to be executed if the current method is aborted. This Builtin will remove the first occurrence of the specified method in the list, starting with the first method added. If

there are multiple occurrences of a specific method, only the first one is removed. Abort methods may not be removed during an abort method.

Lexical structure

```
remove_abort_method(abort_method_name)
```

The lexical elements of the Builtin `remove_abort_method` are shown in Table B.232.

Table B.232 – Builtin `remove_abort_method`

Parameter name	Type	Direction	Usage	Description
<code>abort_method_name</code>	reference	I	m	The ID of the method.
<return>	int	O	m	One of the return codes specified in Table B.347.

B.233 Builtin `remove_abort_method` (version B)

Purpose

The Builtin `remove_abort_method` moves a method from the abort method list. The abort method list contains the methods that are to be executed if the primary method aborts.

This Builtin removes the first occurrence of the method that it finds in the list, starting at the front of the list and moving to the back. The front of the list has the abort methods that were added first (FIFO). If the method occurs more than once in the list, the subsequent occurrences are not removed.

The `method_id` parameter is specified by the method writer using the Builtins `ITEM_ID` and the name of the method. Abort methods cannot call `remove_abort_method`.

Lexical structure

```
remove_abort_method(method_id)
```

The lexical elements of the Builtin `remove_abort_method` are shown in Table B.233.

Table B.233 – Builtin `remove_abort_method`

Parameter name	Type	Direction	Usage	Description
<code>method_id</code>	unsigned long	I	m	The ID of the method.
<return>	long	O	m	One of the return codes specified in Table B.346

B.234 Builtin `remove_all_abort_methods`

Purpose

The Builtin `remove_all_abort_methods` removes all methods from the abort method list. This Builtin is similar to the Builtin `remove_abort_method` or `remove_abort_method` except it removes all of the abort methods. An abort method cannot call `remove_all_abort_methods`.

Lexical structure

```
remove_all_abort_methods(VOID)
```

The lexical element of the Builtin `remove_all_abort_methods` is shown in Table B.234.

Table B.234 – Builtin `remove_all_abort_methods`

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.235 Builtin resolve_array_ref

Purpose

The Builtin resolve_array_ref is used by method developers when parsing things such as item_array_name. The Builtin resolve_array_ref takes the item ID of the REFERENCE_ARRAY and an index and resolves them into the item ID of the REFERENCE_ARRAY element referenced. This Builtin can be used to fill arrays that are passed to display Builtins.

The parameters id and member_id are specified using the Builtins ITEM_ID and MEMBER_ID and the name or reference of the variable in question.

If a member of a REFERENCE_ARRAY has a conditional dependency (that is, the definition of an element is dependent on another value), the dependencies are taken into account during the resolution.

Lexical structure

```
resolve_array_ref(id, member_id)
```

The lexical elements of the Builtin resolve_array_ref are shown in Table B.235.

Table B.235 – Builtin resolve_array_ref

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	m	The item ID of the REFERENCE_ARRAY.
member_id	unsigned long	I	m	The index of the requested REFERENCE_ARRAY element.
<return>	unsigned long	O	m	The item ID of the resolved reference, or zero for an error. In the case of an error, Builtin get_resolve_status may be used to access the actual error code.

B.236 Builtin resolve_block_ref

Purpose

The Builtin resolve_block_ref is used by method developers when parsing things such as BLOCK_A.name. The Builtin resolve_block_ref takes the member ID of a member of a block's characteristic record and returns the item ID for the resolved element.

The parameter member_id is specified using the MEMBER_ID Builtin and the name or reference of the VARIABLE in question. If a member of the characteristics record has a conditional dependency (that is, the definition of an element is dependent on another value), the dependencies are taken into account during the resolution.

Resolves the member_id reference generated for a BLOCK_A element into an item ID, which is then used in Builtin calls.

Lexical structure

```
resolve_block_ref(member_id)
```

The lexical elements of the Builtin resolve_block_ref are shown in Table B.236.

Table B.236 – Builtin resolve_block_ref

Parameter name	Type	Direction	Usage	Description
member_id	unsigned long	I	m	The member ID of BLOCK_A.
<return>	unsigned long	O	m	The item ID of the resolved reference, or zero for an error. In the case of an error, Builtin get_resolve_status may be used to access the actual error code.

B.237 Builtin resolve_list_ref

Purpose

The Builtin resolve_list_ref returns the type of a LIST.

Lexical structure

```
resolve_list_ref(list_id)
```

The lexical elements of the Builtin resolve_list_ref are shown in Table B.237.

Table B.237 – Builtin resolve_list_ref

Parameter name	Type	Direction	Usage	Description
list_id	unsigned long	I	m	The member ID of the LIST.
<return>	unsigned long	O	m	The item ID of the resolved reference, or zero for an error. In the case of an error, Builtin get_resolve_status may be used to access the actual error code.

B.238 Builtin resolve_local_ref

Purpose

The Builtin resolve_local_ref is used by method developers when parsing things such as LOCAL_PARAM.name. It takes the member ID of a LOCAL_PARAMETERS element and returns the item ID of the completely resolved element.

The parameter member_id is specified using the MEMBER_ID Builtin and the name or reference of the variable in question. It resolves the member_id reference generated for a LOCAL_PARAMETERS element into an item ID, which is then used in Builtin calls.

Lexical structure

```
resolve_local_ref(member_id)
```

The lexical elements of the Builtin resolve_local_ref are shown in Table B.238.

Table B.238 – Builtin resolve_local_ref

Parameter name	Type	Direction	Usage	Description
member_id	unsigned long	I	m	The member ID of the local parameter.
<return>	unsigned long	O	m	The item ID of the resolved reference, or zero for an error. In the case of an error, Builtin get_resolve_status may be used to access the actual error code.

B.239 Builtin resolve_local_ref2

Purpose

The Builtin resolve_local_ref2 is used by method developers when parsing things such as block_name[n].LOCAL_PARAM.name. It takes the member ID of a BLOCK_A, the occurrence number of a BLOCK_A, the member ID of a LOCAL_PARAMETERS element and returns the item ID of the completely resolved element.

The parameters block_id and member_id are specified using the MEMBER_ID Builtin. It resolves the member_id reference generated for a LOCAL_PARAMETERS element into an item ID, which is then used in Builtin calls.

Lexical structure

```
resolve_local_ref2(block_id, block_instance, member_id)
```

The lexical elements of the Builtin resolve_local_ref2 are shown in Table B.239.

Table B.239 – Builtin resolve_local_ref2

Parameter name	Type	Direction	Usage	Description
block_id	unsigned long	I	m	The item ID of the BLOCK_A being resolved.
block_instance	unsigned long	I	m	The occurrence number of the BLOCK_A being resolved.
member_id	unsigned long	I	m	The member ID of the local parameter.
<return>	unsigned long	O	m	The item ID of the resolved reference, or zero for an error. In the case of an error, Builtin get_resolve_status may be used to access the actual error code.

B.240 Builtin resolve_param_list_ref

Purpose

The Builtin resolve_param_list_ref is used by method developers when parsing things such as PARAM_LIST name. It takes the member ID of a PARAMETER_LIST element and returns the item ID of the completely resolved element.

The parameter member_id is specified using the MEMBER_ID Builtin and the name or reference of the variable in question. If a member of the parameter list has a conditional dependency-the definition of an element is dependent on another value – the dependencies are taken into account during the resolution.

The Builtin resolve_param_list_ref resolves the member_id reference generated for a PARAMETER_LIST element into an item ID, which is then used in Builtin calls.

Lexical structure

```
resolve_param_list_ref(member_id)
```

The lexical elements of the Builtin resolve_param_list_ref are shown in Table B.240.

Table B.240 – Builtin resolve_param_list_ref

Parameter name	Type	Direction	Usage	Description
member_id	unsigned long	I	m	The member ID of the parameter list.
<return>	unsigned long	O	m	The item ID of the resolved reference, or zero for an error. In the case of an error, Builtin get_resolve_status may be used to access the actual error code.

B.241 Builtin resolve_param_ref

Purpose

The Builtin resolve_param_ref is used by method developers when parsing things such as PARAM.name. It takes the member ID of a PARAMETER element and returns the item ID of the completely resolved element.

The parameter member_id is specified using the MEMBER_ID Builtin and the name or reference of the variable in question. It resolves the member_id reference generated for a PARAMETER element into an item ID, which is then used in Builtin calls.

Lexical structure

```
resolve_param_ref(member_id)
```

The lexical elements of the Builtin `resolve_param_ref` are shown in Table B.241.

Table B.241 – Builtin `resolve_param_ref`

Parameter name	Type	Direction	Usage	Description
member_id	unsigned long	I	m	The member ID of the parameter.
<return>	unsigned long	O	m	The item ID of the resolved reference, or zero for an error. In the case of an error, Builtin <code>get_resolve_status</code> may be used to access the actual error code.

B.242 Builtin `resolve_param_ref2`

Purpose

The Builtin `resolve_param_ref2` is used by method developers when parsing things such as `block_name[n].PARAM.name`. It takes the member ID of a `BLOCK_A`, an occurrence number, and a `PARAMETER` element and returns the item ID of the completely resolved element.

The block name and parameter member_id is specified using the `MEMBER_ID` Builtin and the name or reference of the variable in question. It resolves the member_id reference generated for a `PARAMETER` element into an item ID, which is then used in Builtin calls.

Lexical structure

```
resolve_param_ref2(block_id, block_instance, member_id)
```

The lexical elements of the Builtin `resolve_param_ref2` are shown in Table B.242.

Table B.242 – Builtin `resolve_param_ref2`

Parameter name	Type	Direction	Usage	Description
block_id	unsigned long	I	m	The item ID of the <code>BLOCK_A</code> being resolved.
block_instance	unsigned long	I	m	The occurrence number of the <code>BLOCK_A</code> being resolved.
member_id	unsigned long	I	m	The member ID of the parameter.
<return>	unsigned long	O	m	The item ID of the resolved reference, or zero for an error. In the case of an error, Builtin <code>get_resolve_status</code> may be used to access the actual error code.

B.243 Builtin `resolve_record_ref`

Purpose

The Builtin `resolve_record_ref` is used by method developers when parsing things such as `record_name.member_name`. It takes the item ID of a record and the member ID of a record element or member and returns the item ID of the completely resolved element.

If a member of a record has a conditional dependency (that is, the definition of an element is dependent on another value), the dependencies are taken into account during the resolution.

The parameters `id` and `member_id` are specified using the Builtins `ITEM_ID` and `MEMBER_ID` and the name or reference of the variable in question.

Resolves the member_id reference generated for a `RECORD` element into an item ID, which is then used in Builtin calls.

Lexical structure

```
resolve_record_ref(id, member_id)
```

The lexical elements of the Builtin `resolve_record_ref` are shown in Table B.243.

Table B.243 – Builtin `resolve_record_ref`

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	m	The item ID of the RECORD being resolved.
member_id	unsigned long	I	m	The member ID of the record member.
<return>	unsigned long	O	m	The item ID of the resolved reference, or zero for an error. In the case of an error, Builtin <code>get_resolve_status</code> may be used to access the actual error code.

B.244 Builtin `resolve_record_ref2`

Purpose

The Builtin `resolve_record_ref2` is used by method developers when parsing things such as `record_name.member_name` of a specific block. It takes the item ID of a block, the occurrence number of a block, the item ID of a record and the member ID of a record element or member and returns the item ID of the completely resolved element.

If a member of a record has a conditional dependency (that is, the definition of an element is dependent on another value), the dependencies are taken into account during the resolution.

The parameters `block_id`, `id` and `member_id` are specified using the Builtins `ITEM_ID` and `MEMBER_ID` and the name or reference of the variable in question.

Resolves the `member_id` reference generated for a RECORD element into an item ID, which is then used in Builtin calls.

Lexical structure

```
resolve_record_ref2(block_id, block_instance, id, member_id)
```

The lexical elements of the Builtin `resolve_record_ref2` are shown in Table B.244.

Table B.244 – Builtin `resolve_record_ref2`

Parameter name	Type	Direction	Usage	Description
block_id	unsigned long	I	m	The item ID of the BLOCK_A being resolved.
block_instance	unsigned long	I	m	The occurrence number of the BLOCK_A being resolved.
id	unsigned long	I	m	The item ID of the RECORD being resolved.
member_id	unsigned long	I	m	The member ID of the record member.
<return>	unsigned long	O	m	The item ID of the resolved reference, or zero for an error. In the case of an error, Builtin <code>get_resolve_status</code> may be used to access the actual error code.

B.245 Builtin `ret_double_value`

Purpose

The Builtin `ret_double_value` calls the Builtin `get_double_value` and returns the value.

Lexical structure

```
ret_double_value(item_id, member_id)
```

The lexical elements of the Builtin `ret_double_value` are shown in Table B.245.

Table B.245 – Builtin `ret_double_value`

Parameter name	Type	Direction	Usage	Description
item_id	unsigned long	I	m	The item id of the VARIABLE.
member_id	unsigned long	I	m	The member of the VARIABLE.
<return>	double	O	m	The specified value.

B.246 Builtin `ret_double_value2`**Purpose**

The Builtin `ret_double_value2` calls the Builtin `get_double_value2` and returns the value.

Lexical structure

```
ret_double_value2(block_id, block_instance, item_id, member_id)
```

The lexical elements of the Builtin `ret_double_value2` are shown in Table B.246.

Table B.246 – Builtin `ret_double_value2`

Parameter name	Type	Direction	Usage	Description
block_id	unsigned long	I	m	The item id of the block to be accessed.
block_instance	unsigned long	I	m	The occurrence number of the block to be accessed.
item_id	unsigned long	I	m	The item id of the VARIABLE.
member_id	unsigned long	I	m	The member of of the VARIABLE.
<return>	double	O	m	The specified value.

B.247 Builtin `ret_float_value`**Purpose**

The Builtin `ret_float_value` calls the Builtin `get_float_value` and returns the value.

Lexical structure

```
ret_float_value(item_id, member_id)
```

The lexical elements of the Builtin `ret_float_value` are shown in Table B.247.

Table B.247 – Builtin `ret_float_value`

Parameter name	Type	Direction	Usage	Description
item_id	unsigned long	I	m	The item id of the VARIABLE.
member_id	unsigned long	I	m	The member of the VARIABLE.
<return>	float	O	m	The specified value.

B.248 Builtin ret_float_value2

Purpose

The Builtin ret_float_value2 calls the Builtin get_float_value2 and returns the value.

Lexical structure

```
ret_float_value2(block_id, block_instance, item_id, member_id)
```

The lexical elements of the Builtin ret_float_value2 are shown in Table B.248.

Table B.248 – Builtin ret_float_value2

Parameter name	Type	Direction	Usage	Description
block_id	unsigned long	I	m	The item id of the block to be accessed.
block_instance	unsigned long	I	m	The occurrence number of the block to be accessed.
item_id	unsigned long	I	m	The item id of the VARIABLE.
member_id	unsigned long	I	m	The member of the VARIABLE.
<return>	float	O	m	The specified value.

B.249 Builtin ret_signed_value

Purpose

The Builtin ret_signed_value calls the Builtin get_signed_value and returns the value.

Lexical structure

```
ret_signed_value(item_id, member_id)
```

The lexical elements of the Builtin ret_signed_value are shown in Table B.249.

Table B.249 – Builtin ret_signed_value

Parameter name	Type	Direction	Usage	Description
item_id	unsigned long	I	m	The item id of the VARIABLE.
member_id	unsigned long	I	m	The member of the VARIABLE.
<return>	long	O	m	The specified value.

B.250 Builtin ret_signed_value2

Purpose

The Builtin ret_signed_value2 calls the Builtin get_signed_value2 and returns the value.

Lexical structure

```
ret_signed_value2(block_id, block_instance, item_id, member_id)
```

The lexical elements of the Builtin ret_signed_value2 are shown in Table B.250.

Table B.250 – Builtin ret_signed_value2

Parameter name	Type	Direction	Usage	Description
block_id	unsigned long	I	m	The item id of the block to be accessed.

Parameter name	Type	Direction	Usage	Description
block_instance	unsigned long	I	m	The occurrence number of the block to be accessed.
item_id	unsigned long	I	m	The item id of the VARIABLE.
member_id	unsigned long	I	m	The member of the VARIABLE.
<return>	long	O	m	The specified value.

B.251 Builtin ret_unsigned_value

Purpose

The Builtin ret_unsigned_value calls the Builtin get_unsigned_value and returns the value.

Lexical structure

ret_unsigned_value(item_id, member_id)

The lexical elements of the Builtin ret_unsigned_value are shown in Table B.251.

Table B.251 – Builtin ret_unsigned_value

Parameter name	Type	Direction	Usage	Description
item_id	unsigned long	I	m	The item id of the VARIABLE.
member_id	unsigned long	I	m	The member of the VARIABLE.
<return>	unsigned long	O	m	The specified value.

B.252 Builtin ret_unsigned_value2

Purpose

The Builtin ret_unsigned_value2 calls the Builtin get_unsigned_value2 and returns the value.

Lexical structure

ret_unsigned_value2(block_id, block_instance, item_id, member_id)

The lexical elements of the Builtin ret_unsigned_value2 are shown in Table B.252.

Table B.252 – Builtin ret_unsigned_value2

Parameter name	Type	Direction	Usage	Description
block_id	unsigned long	I	m	The item id of the block to be accessed.
block_instance	unsigned long	I	m	The occurrence number of the block to be accessed.
item_id	unsigned long	I	m	The item id of the VARIABLE.
member_id	unsigned long	I	m	The member of the VARIABLE.
<return>	unsigned long	O	m	The specified value.

B.253 Builtin retry_on_all_comm_errors

Purpose

The Builtin `retry_on_all_comm_errors` causes the last request made by a method to be tried again after any communications error. A request will be retried three times before it fails and an error return is returned.

This Builtin frees a method from having to explicitly retry a request after any communications errors.

The communication error codes are defined in the standard profiled and their related strings are defined in the text dictionary.

NOTE A communications error of zero is not considered a communications error.

Lexical structure

`retry_on_all_comm_errors(VOID)`

The lexical element of the Builtin `retry_on_all_comm_errors` is shown in Table B.253.

Table B.253 – Builtin `retry_on_all_comm_errors`

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.254 Builtin RETRY_ON_ALL_COMM_STATUS

Purpose

The Builtin `RETRY_ON_ALL_COMM_STATUS` will set all of the bits in the comm status retry mask. This will cause the system to retry the current command if the device returns any comm status value. Comm status is defined to be the first data octet returned in a transaction, when bit 7 of this octet is set.

The retry and abort masks are reset to their default values at the start of each method. so the new mask value will only be valid during the current method. See the `send_command` function for implementation of the masks. See Builtin `ABORT_ON_RESPONSE_CODE` for a list of the available masks, and their default values.

The mask affected by this function is used by the Builtins `send`, `send_trans`, `send_command`, `send_command_trans`, `ext_send_command`, `ext_send_command_trans`, `get_more_status` and `display`.

Lexical structure

`RETRY_ON_ALL_COMM_STATUS(VOID)`

The lexical element of the Builtin `RETRY_ON_ALL_COMM_STATUS` is shown in Table B.254.

Table B.254 – Builtin `RETRY_ON_ALL_COMM_STATUS`

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.255 Builtin RETRY_ON_ALL_DEVICE_STATUS

Purpose

The Builtin `RETRY_ON_ALL_DEVICE_STATUS` will set all of the bits in the device status retry mask. This will cause the system to retry the current command if the device returns any device status value. Device status is defined to be the second data octet returned in a transaction.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method.

The mask affected by this function is used by the Builtins send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status and display.

Lexical structure

RETRY_ON_ALL_DEVICE_STATUS (VOID)

The lexical element of the Builtin RETRY_ON_ALL_DEVICE_STATUS is shown in Table B.255.

Table B.255 – Builtin RETRY_ON_ALL_DEVICE_STATUS

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.256 Builtin RETRY_ON_ALL_RESPONSE_CODES

Purpose

The Builtin RETRY_ON_ALL_RESPONSE_CODES will set all of the bits in the response code retry mask. This will cause the System to retry the current command if the device returns any code value.

The response code is defined to be the first data octet returned in a transaction, when bit 7 of this octet is 0.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks, and their default values.

The mask affected by this function is used by the Builtins send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status and display.

Lexical structure

RETRY_ON_ALL_RESPONSE_CODES (VOID)

The lexical element of the Builtin RETRY_ON_ALL_RESPONSE_CODES is shown in Table B.256.

Table B.256 – Builtin RETRY_ON_ALL_RESPONSE_CODES

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.257 Builtin retry_on_all_response_codes

Purpose

The Builtin retry_on_all_response_codes causes the last request made by a method to be retried after any response code. A request will be retried three times before it fails and an error return is returned. This is the default action by the Builtins when a method is just started.

This Builtin frees a method from having to explicitly retry a request after any response code.

A response code is an integer value representing an application-specific error condition.

NOTE A response code value of zero is not considered an error response code. Each response code and response code type is defined in the EDD.

Lexical structure

```
retry_on_all_response_codes(void)
```

The lexical element of the Builtin `retry_on_all_response_codes` is shown in Table B.257.

Table B.257 – Builtin `retry_on_all_response_codes`

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.258 Builtin `RETRY_ON_COMM_ERROR`

Purpose

The Builtin `RETRY_ON_COMM_ERROR` will set the no comm error mask such that the current command will be retried if a comm error is found while sending the command.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the `send_command` function for implementation of the masks. See Builtin `ABORT_ON_RESPONSE_CODE` for a list of the available masks and their default values.

The mask affected by this function is used by the Builtins `send`, `send_trans`, `send_command`, `send_command_trans`, `ext_send_command`, `ext_send_command_trans`, `get_more_status` and `display`.

Lexical structure

```
RETRY_ON_COMM_ERROR(VOID)
```

The lexical element of the Builtin `RETRY_ON_COMM_ERROR` is shown in Table B.258.

Table B.258 – Builtin `RETRY_ON_COMM_ERROR`

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.259 Builtin `retry_on_comm_error`

Purpose

The Builtin `retry_on_comm_error` causes the last request made by a method to be retried after a specified communications error. A request will be retried three times before it fails and an error return is returned.

This Builtin frees a method from having to explicitly retry a request after a given communications error is received.

The communications error codes are defined by the consortia. Their related strings may be defined in a text dictionary. A communications error of zero is not considered a communications error.

Lexical structure

```
retry_on_comm_error(error)
```

The lexical elements of the Builtin `retry_on_comm_error` are shown in Table B.259.

Table B.259 – Builtin retry_on_comm_error

Parameter name	Type	Direction	Usage	Description
error	unsigned long	I	m	The communication error.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.260 Builtin RETRY_ON_COMM_STATUS**Purpose**

The Builtin RETRY_ON_COMM_STATUS will set the correct bit(s) in the comm status retry mask such that the specified comm status value will cause the current command to be retried. Comm status is defined to be the first data octet returned in a transaction, when bit 7 of this octet is 1.

The retry and abort masks are result to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_method function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this function is used by the Builtins send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status and display.

Lexical structure

RETRY_ON_COMM_STATUS(comm_status)

The lexical elements of the Builtin RETRY_ON_COMM_STATUS are shown in Table B.260.

Table B.260 – Builtin RETRY_ON_COMM_STATUS

Parameter name	Type	Direction	Usage	Description
comm_status	int	I	m	The new communication status retry mask.
<return>	VOID	—	—	—

B.261 Builtin RETRY_ON_DEVICE_STATUS**Purpose**

The Builtin RETRY_ON_DEVICE_STATUS will set the correct bit(s) in the device status retry mask such that the specified device status value will cause the current command to be retried. Device status is defined to be the second data octet returned in a transaction.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this function is used by the Builtins send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status and display.

Lexical structure

RETRY_ON_DEVICE_STATUS(device_status)

The lexical elements of the Builtin RETRY_ON_DEVICE_STATUS are shown in Table B.261.

Table B.261 – Builtin RETRY_ON_DEVICE_STATUS

Parameter name	Type	Direction	Usage	Description
device_status	int	I	m	The new communication status retry mask.
<return>	VOID	—	—	—

B.262 Builtin RETRY_ON_NO_DEVICE**Purpose**

The Builtin RETRY_ON_NO_DEVICE will set the no device mask such that the current command will be retried if no device is found while sending a command.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this function is used by the Builtins send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status and display.

Lexical structure

RETRY_ON_NO_DEVICE (VOID)

The lexical element of the Builtin RETRY_ON_NO_DEVICE is shown in Table B.262.

Table B.262 – Builtin RETRY_ON_NO_DEVICE

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.263 Builtin RETRY_ON_RESPONSE_CODE**Purpose**

The Builtin RETRY_ON_RESPONSE_CODE will set the correct bit(s) in the response code retry mask such that the specified response code value will cause the current command to be retried. The response code is defined to be the first data octet returned in a transaction, when bit 7 of this octet is 0.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of available masks and their default values.

The mask affected by this function is used by the Builtins send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status and display.

Lexical structure

RETRY_ON_RESPONSE_CODE (response_code)

The lexical elements of the Builtin RETRY_ON_RESPONSE_CODE are shown in Table B.263.

Table B.263 – Builtin RETRY_ON_RESPONSE_CODE

Parameter name	Type	Direction	Usage	Description
response_code	int	I	m	The new communication status retry mask.
<return>	VOID	—	—	—

B.264 Builtin retry_on_response_code**Purpose**

The Builtin `retry_on_response_code` causes the last request made by a method to be retried three times after a specific response code is received.

This Builtin frees a method from having to explicitly retry a request after a given response code is received.

A response code is a value representing an application-specific error condition.

NOTE A response code value of zero is not considered an error response code. Each response code and response code type is defined in the EDD.

Retries the method's last request upon receiving the specified response code.

Lexical structure

`retry_on_response_code (code)`

The lexical elements of the Builtin `retry_on_response_code` are shown in Table B.264.

Table B.264 – Builtin retry_on_response_code

Parameter name	Type	Direction	Usage	Description
code	unsigned long	I	m	The response code being resolved.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.265 Builtin round**Purpose**

The Builtin `round` rounds a floating point value to the nearest integer.

Lexical structure

`round (x)`

The lexical elements of the Builtin `round` are shown in Table B.265.

Table B.265 – Builtin round

Parameter name	Type	Direction	Usage	Description
x	double	I	m	A floating point value.
<return>	double	O	m	The rounded value.

B.266 Builtin save_on_exit**Purpose**

The Builtin `save_on_exit` sets the termination action to save any changes made by the method to variables in the variable table not already sent to the device. The values in the device are not affected. As long as the application has not changed the values, the values are saved as if written by an application.

When a method modifies a variable's value, it is really just changing a resident copy of that value in the EDD application. This change is effective only for the life of the method. To make the values permanent in the variable table, the method shall save the values before exiting.

All values that have been changed, but not sent to the device, are handled at the termination of the method by either sending them to the device, discarding them, or saving the value changes in a way that survives the end of the method (but does not affect the device). Saved values can be used by other methods.

If a method does not call Builtin discard_on_exit, save_on_exit, or send_on_exit prior to exiting, any values in the Builtin are discarded as if discard_on_exit was called.

If the method aborts, then the Builtin discard_on_exit option is enforced even if Builtin save_on_exit were called before the method aborted.

Lexical structure

save_on_exit (VOID)

The lexical element of the Builtin save_on_exit is shown in Table B.266.

Table B.266 – Builtin save_on_exit

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.267 Builtin save_values

Purpose

The Builtin save_values will cause the values of all device variables modified during a method session to remain permanent after the method has been exited.

Device variables that are modified, but are not sent to the connected device, will be restored to their previous value if this function is not called.

Lexical structure

save_values (VOID)

The lexical element of the Builtin save_values is shown in Table B.267.

Table B.267 – Builtin save_values

Parameter name	Type	Direction	Usage	Description
<return>	int	O	m	One of the return codes specified in Table B.347.

B.268 Builtin seconds_to_TIME_VALUE

Purpose

The Builtin seconds_to_TIME_VALUE converts a seconds value to a TIME_VALUE.

Lexical structure

seconds_to_TIME_VALUE (seconds)

The lexical element of the Builtin seconds_to_TIME_VALUE is shown in Table B.268.

Table B.268 – Builtin seconds_to_TIME_VALUE

Parameter name	Type	Direction	Usage	Description
seconds	double	I	m	The number seconds.

Parameter name	Type	Direction	Usage	Description
<return>	unsigned long	O	m	The TIME value.

B.269 Builtin SELECT_FROM_LIST

Purpose

The Builtin SELECT_FROM_LIST has the same functionality as Builtin SELECT_FROM_LIST, except that device variables are not allowed in the prompt string. There shall be at least two options in the select list.

NOTE 1 Previously displayed messages are not guaranteed to remain on the screen.

NOTE 2 Some EDD applications will clear the display upon each call to this function.

Lexical structure

SELECT_FROM_LIST(prompt, option_list)

The lexical elements of the Builtin SELECT_FROM_LIST are shown in Table B.269.

Table B.269 – Builtin SELECT_FROM_LIST

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	A message to be displayed.
option_list	char[]	I	m	The option string, consisting of separated text, that defines the menu items.
<return>	int	O	m	The result of the selection as position (zero based).

B.270 Builtin select_from_list

Purpose

The Builtin select_from_list will display a prompt and a supplied list of options for the user to select from. Once a selection is made, the position of that option in the option list is returned. The position in the list is zero-based. For example, if the list provided contains two items and if the first item is selected, this function will return a 0. If the second item is selected, a 1 is returned.

The prompt and/or list may also contain embedded local and/or device variable values (see put_message for the Lexical Structure).

The options are passed to the function as a single string with semicolon separators between the options. There shall be at least two options in the select list.

NOTE 1 Previously displayed messages are not guaranteed to remain on the screen.

NOTE 2 Some EDD applications will clear the display upon each call to this function.

Lexical structure

select_from_list(prompt, global_var_ids, option_list)

The lexical elements of the Builtin SELECT_FROM_LIST are shown in Table B.270.

Table B.270 – Builtin select_from_list

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	A message to be displayed.
global_var_ids	array of int	I	m	An array which contains unique identifiers of VARIABLE instance.
option_list	char[]	I	m	The option string, consisting of separated text, that defines the menu items.

Parameter name	Type	Direction	Usage	Description
<return>	int	O	m	The result of the selection as position (zero based).

B.271 Builtin select_from_menu

Purpose

The Builtin select_from_menu displays a list of options or menu items for the user to select from and retrieves the selection number. When the user has made a choice it is returned to the calling method. Selection numbers start at 1 and range up to the number of options specified by the parameter options. Variable values may be embedded in the prompt message string with formatting information. The option string defines the menu from which the user selects and is a string consisting of the selection options; each option is separated from the other options with a semi-colon. Each option may have language codes embedded in it.

- The application displays the strings in accordance with the following guidelines:
- the menu items appear on the screen in the same order as listed in the string;
- the prompt string appears at the top of the menu display on the screen;
- the prompt string is not selectable;
- the menu is presented vertically.

Lexical structure

```
select_from_menu(prompt, ids, indices, id_count, options, selection)
```

The lexical elements of the Builtin select_from_menu are shown in Table B.271.

Table B.271 – Builtin select_from_menu

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	A message to be displayed.
ids	array of unsigned long	I	m	The item IDs of the VARIABLES used in the prompt message.
indices	array of unsigned long	I	m	The member IDs or indices of the VARIABLES used in the prompt message.
id_count	long	I	m	The number of REFERENCE_ARRAY elements in the REFERENCE_ARRAY for ids and indices. Set to zero if the REFERENCE_ARRAY for ids and indices are zero.
options	char[]	I	m	The option string, consisting of separated text, that defines the menu items.
selection	long	O	m	The result of the selection as position (zero based).
<return>	long	O	m	One of the return codes specified in Table B.346.

B.272 Builtin select_from_menu2

Purpose

The Builtin select_from_menu2 displays a list of options or menu items for the user to select from and retrieves the selection number. When the user has made a choice it is returned to the calling method. Selection numbers start at 1 and range up to the number of options specified by the parameter options. Variable values may be embedded in the prompt message string with formatting information. The option string defines the menu from which the user selects and is a string consisting of the selection options; each option is separated from the other options with a semi-colon. Each option may have language codes embedded in it.

- The application displays the strings in accordance with the following guidelines:

- the menu items appear on the screen in the same order as listed in the string;
- the prompt string appears at the top of the menu display on the screen;
- the prompt string is not selectable;
- the menu is presented vertically.

Lexical structure

```
select_from_menu2(prompt, blk_ids, blk_nums, ids, indices, id_count,
options, selection)
```

The lexical elements of the Builtin `select_from_menu2` are shown in Table B.272.

Table B.272 – Builtin `select_from_menu2`

Parameter name	Type	Direction	Usage	Description
prompt	char[]	l	m	A message to be displayed.
blk_ids	array of unsigned long	l	m	The item IDs of the BLOCK_A instances used in the prompt message.
blk_nums	array of unsigned long	l	m	The occurrence numbers of the BLOCK_A instances used in the prompt message.
ids	array of unsigned long	l	m	The item IDs of the VARIABLES used in the prompt message.
indices	array of unsigned long	l	m	The member IDs or indices of the VARIABLES used in the prompt message.
id_count	long	l	m	The number of REFERENCE_ARRAY elements in the REFERENCE_ARRAY for ids and indices. Set to zero if the REFERENCE_ARRAY for ids and indices are zero.
options	char[]	l	m	The option string, consisting of separated text, that defines the menu items.
selection	long	o	m	The result of the selection as position (zero based).
<return>	long	o	m	One of the return codes specified in Table B.346.

B.273 Builtin `send`

Purpose

The Builtin `send` will send the specified command and return the response code, communications status, and command status respectively in the `cmd_status` array.

The octets in the `cmd_status` array are logically AND with the appropriate abort masks to determine if the method should be aborted. If there is a match, the method is aborted and the abort methods (if any were designated) are run. If there is no match for the abort masks, the same data is AND with the appropriate retry masks. If there is a match, the command will be retried. This continues until either the command succeeds, an abort occurs, or the maximum number of retries is reached.

The status octets for the command are returned in the `cmd_status` parameter, a pointer to a 3-octet array, which is allocated in the method making the call.

Lexical structure

```
send(cmd_number, cmd_status)
```

The lexical elements of the Builtin `send` are shown in Table B.273.

Table B.273 – Builtin send

Parameter name	Type	Direction	Usage	Description
cmd_number	int	I	m	The number of command to send.
cmd_status	char*	O	m	The status octets of the send command.
<return>	int	O	m	One of the return codes specified in Table B.347.

B.274 Builtin send_all_values

Purpose

The Builtin send_all_values sends all of the values in the variable table that the method modified but has not previously sent to the device. This action changes the device values to match the values set by the method.

This Builtin does nothing for a non-caching EDD application because it does not have a variable table. Updates the device with all of the values in the variable table that the method has changed.

Lexical structure

send_all_values(VOID)

The lexical element of the Builtin send_all_values is shown in Table B.274.

Table B.274 – Builtin send_all_values

Parameter name	Type	Direction	Usage	Description
<return>	long	O	m	One of the return codes specified in Table B.346.

B.275 Builtin send_command

Purpose

The Builtin send_command will send the specified command. If the status bit indicating that more data is available is returned (status class MORE), command 48 is automatically issued. The status/data octets from the original command and command 48 (if it was sent) are then logically AND with the appropriate abort masks to determine if the method should be aborted. If there is a match, the method is aborted and the abort methods (if any were designated) are run. If there is no match for the abort masks, the same data is AND with the appropriate retry masks. If there is a match, the appropriate command will be retried (either the specified command, or command 48). This continues until either the command succeeds, an abort occurs, or the maximum number of retries is reached. No status or data octets are returned with this function.

Lexical structure

send_command(cmd_number)

The lexical elements of the Builtin send_command are shown in Table B.275.

Table B.275 – Builtin send_command

Parameter name	Type	Direction	Usage	Description
cmd_number	int	I	m	The number of command to send.
<return>	int	O	m	One of the return codes specified in Table B.347.

B.276 Builtin send_command_trans

Purpose

The Builtin send_command_trans sends the command with the specified transaction to the device. This function is to be used to send commands that have been defined with multiple transactions.

The Builtin send_command_trans will send the specified command. If the status bit indicating that more data is available is returned (status class MORE), command 48 is automatically issued. The status/data octets from the original command and command 48 (if it was sent) are then logically ANDed with the appropriate abort masks to determine if the method should be aborted. If there is a match, the method for the abort masks, the same data is ANDed with the appropriate retry masks. If there is a match, the appropriate command will be retried (either the specified command, or command 48). This continues until either the command succeeds, an abort occurs, or the maximum number of retries is reached. No status or data octets are returned with this command.

Lexical structure

```
send_command_trans(cmd_number, transaction)
```

The lexical elements of the Builtin send_command_trans are shown in Table B.276.

Table B.276 – Builtin send_command_trans

Parameter name	Type	Direction	Usage	Description
cmd_number	int	I	m	The number of command to send.
transaction	int	I	m	The number of the TRANSACTION.
<return>	int	O	m	One of the return codes specified in Table B.347.

B.277 Builtin send_on_exit

Purpose

The Builtin send_on_exit sets the termination action to send any changes made by the method to variables stored in the variable table of the EDD application, which have not already been sent to the device. This Builtin does nothing in a non-caching EDD application, because a non-caching EDD application does not have a variable table.

When a method modifies a variable value, it changes a copy of that value. This change is effective only for the life of the method. In order to make the change in the device, the method shall send the value(s) to the device or call this Builtin before the method exits.

All values that are changed, but not sent to the device, are handled at the termination of the method by either sending them to the device, discarding them, or saving the value changes in a way that survives the end of the method. The changed values in the variable table are sent after the method terminates.

If a method does not call Builtin discard_on_exit, or Builtin send_on_exit prior to the exiting, any values changed by the Builtin are discarded as if Builtin discard_on_exit was called. If the method aborts then the Builtin discard_on_exit option is enforced, even if Builtin save_on_exit, or Builtin send_on_exit were called before the method aborted.

Lexical structure

```
send_on_exit(VOID)
```

The lexical element of the Builtin send_on_exit is shown in Table B.277.

Table B.277 – Builtin send_on_exit

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.278 Builtin send_trans**Purpose**

The Builtin send_trans sends a command with the specified transaction to the device. This function is to be used to send command that have been defined with multiple transactions.

The Builtin send_trans will send the specified command and return the response code, communications status, and command status respectively in the cmd_status array. These octets are logically ANDed with the appropriate abort masks to determine if the method should be aborted. If there is a match, the method is aborted and the abort methods (if any were designated) are run. If there is a match, the command will be retried. This continues until either the command succeeds, an abort occurs, or the maximum number of retries is reached.

The status octets for the command are returned in the cmd_status parameter, a 3-octet array, allocated in the method making the call.

Lexical structure

```
send_trans(cmd_number, transaction, cmd_status)
```

The lexical elements of the Builtin send_trans are shown in Table B.278.

Table B.278 – Builtin send_trans

Parameter name	Type	Direction	Usage	Description
cmd_number	int	I	m	The number of command to send.
transaction	int	I	m	The number of the TRANSACTION.
cmd_status	char*	O	m	The status octets of the send command.
<return>	int	O	m	One of the return codes specified in Table B.347.

B.279 Builtin send_value**Purpose**

The Builtin send_value causes the value of the specked variable in the variable table to be sent to the device, thus updating the device value. This Builtin does nothing in a non-caching EDD application, because a non-caching EDD application does not have a variable table. Some values are READ_ONLY in the device and cannot be written. Attempting to do so will cause a Builtin error to be returned.

The parameters id and member id are specified using the Builtins ITEM_ID and MEMBER_ID and the name or reference of the variable in question.

Lexical structure

```
send_value(id, member_id)
```

The lexical elements of the Builtin send_value are shown in Table B.279.

Table B.279 – Builtin send_value

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	m	The item ID of the VARIABLE to read.

Parameter name	Type	Direction	Usage	Description
member_id	unsigned long	I	m	The member ID of the VARIABLE to read.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.280 Builtin send_value2

Purpose

The Builtin send_value2 causes the value of the specked variable in the variable table to be sent to the device, thus updating the device value. This Builtin does nothing in a non-caching EDD application, because a non-caching EDD application does not have a variable table. Some values are READ_ONLY in the device and cannot be written. Attempting to do so will cause a Builtin error to be returned.

The parameters id and member id are specified using the Builtins ITEM_ID and MEMBER_ID and the name or reference of the variable in question.

Lexical structure

```
send_value2(blk_id, blk_num, id, member_id)
```

The lexical elements of the Builtin send_value2 are shown in Table B.280.

Table B.280 – Builtin send_value2

Parameter name	Type	Direction	Usage	Description
blk_id	unsigned long	I	m	The item ID of the BLOCK_A instance to access.
blk_num	unsigned long	I	m	The occurrence number of the BLOCK_A instance to access.
id	unsigned long	I	m	The item ID of the VARIABLE to read.
member_id	unsigned long	I	m	The member ID of the VARIABLE to read.
<return>	long	O	m	One of the return codes specified in Table B.346.

B.281 Builtin SET_NUMBER_OF_RETRIES

Purpose

The Builtin SET_NUMBER_OF_RETRIES sets up the number of times a command will be retried due to the appropriate retry masks. This value is defaulted to 0 at the beginning of each method.

Lexical structure

```
SET_NUMBER_OF_RETRIES(n)
```

The lexical elements of the Builtin SET_NUMBER_OF_RETRIES are shown in Table B.281.

Table B.281 – Builtin SET_NUMBER_OF_RETRIES

Parameter name	Type	Direction	Usage	Description
n	int	I	m	The number of retries of a command.
<return>	int	O	m	One of the return codes specified in Table B.347.

B.282 Builtin sgetval

Purpose

The Builtin sgetval gets the value of a string type (ASCII, PACKED_ASCII, PASSWORD, EUC) DD variable. It gets the value of the variable from which the method is called as an action (e.g. PRE_EDIT_ACTIONS).

Lexical Structure

sgetval (value)

The lexical elements of the Builtin sgetval are shown in Table B.282.

Table B.282 – Builtin sgetval

Parameter name	Type	Direction	Usage	Description
value	DD_STRING	O	m	The value of the VARIABLE.
<return>	int	O	m	One of the return codes specified in Table B.347.

B.283 Builtin ShortToByte

Purpose

The Builtin ShortToByte will provide a vector of unsigned char values representing the particular elements of a short value.

Lexical structure

ShortToByte(in, out1, out2)

The lexical element of the Builtin ShortToByte is shown in Table B.283.

Table B.283 – Builtin ShortToByte

Parameter name	Type	Direction	Usage	Description
in	short	I	m	The short value to convert.
out1	unsigned char	O	m	First element of the short value (most significant byte).
out2	unsigned char	O	m	Second element of the short value (least significant byte).
<return>	VOID	—	—	—

B.284 Builtin sin

Purpose

The Builtin sin returns the sine a floating point value.

Lexical structure

sin(x)

The lexical elements of the Builtin sin are shown in Table B.284.

Table B.284 – Builtin sin

Parameter name	Type	Direction	Usage	Description
x	double	I	m	A floating-point value.
<return>	double	O	m	The sine.

B.285 Builtin sinh

Purpose

The Builtin sinh returns the hyperbolic sine of a floating point value.

Lexical structure

`sinh(x)`

The lexical elements of the Builtin sinh are shown in Table B.285.

Table B.285 – Builtin sinh

Parameter name	Type	Direction	Usage	Description
x	double	I	m	A floating point value.
<return>	double	O	m	The hyperbolic sine.

B.286 Builtin sqrt

Purpose

The Builtin sqrt returns the square root of a floating-point value.

Lexical structure

`sqrt(x)`

The lexical elements of the Builtin sqrt are shown in Table B.286.

Table B.286 – Builtin sqrt

Parameter name	Type	Direction	Usage	Description
x	double	I	m	A floating-point value.
<return>	double	O	m	The square root.

B.287 Builtin ssetval

Purpose

The Builtin ssetval sets the value of a string type (ASCII, PACKED_ASCII, PASSWORD, EUC) DD variable. It sets the value of the variable from which the method was called as an action (e.g. PRE_EDIT_ACTIONS).

Lexical Structure

`ssetval(value)`

The lexical elements of the Builtin ssetval are shown in Table B.287.

Table B.287 – Builtin ssetval

Parameter name	Type	Direction	Usage	Description
value	DD_STRING	I	m	The value of the VARIABLE.
<return>	DD_STRING	O	m	The value of the VARIABLE.

B.288 Builtin strcmp

Purpose

The Builtin strcmp compare two strings.

Lexical structure

```
strcmp(string1, string2)
```

The lexical elements of the Builtin strcmp are shown in Table B.288.

Table B.288 – Builtin strcmp

Parameter name	Type	Direction	Usage	Description
string1	DD_STRING	I	m	The first string.
string2	DD_STRING	I	m	The second string.
<return>	int	O	m	Zero if string1 and string2 are the same, returns a negative value if string1 is less than string2, returns a positive value if string1 is greater than string2.

B.289 Builtin strleft**Purpose**

The Builtin strleft extracts letters from the beginning of a string.

Lexical structure

```
strleft(string, length)
```

The lexical elements of the Builtin strleft are shown in Table B.289.

Table B.289 – Builtin strleft

Parameter name	Type	Direction	Usage	Description
string	DD_STRING	I	m	The input string.
length	int	I	m	The number of characters to be extracted from the input string.
<return>	DD_STRING	O	m	The specified substring.

B.290 Builtin strlen**Purpose**

The Builtin strlen returns the length of a string.

Lexical structure

```
strlen(string)
```

The lexical elements of the Builtin strlen are shown in Table B.290.

Table B.290 – Builtin strlen

Parameter name	Type	Direction	Usage	Description
string	DD_STRING	I	m	The string whose length is to be returned.
<return>	int	O	m	The length of the string.

B.291 Builtin strlwr**Purpose**

The Builtin strlwr returns a lower-case version of string.

Lexical structure

```
strlwr(string)
```


The lexical elements of the Builtin `strlwr` are shown in Table B.291.

Table B.291 – Builtin `strlwr`

Parameter name	Type	Direction	Usage	Description
string	DD_STRING	I	m	The string to be converted.
<return>	DD_STRING	O	m	A lower-case version of the string.

B.292 Builtin `strmid`

Purpose

The Builtin `strmid` returns a substring of a string.

Lexical structure

`strmid(string, position, length)`

The lexical elements of the Builtin `strmid` are shown in Table B.292.

Table B.292 – Builtin `strmid`

Parameter name	Type	Direction	Usage	Description
string	DD_STRING	I	m	The input string.
position	int	I	m	The first character to be extracted from the input string.
length	int	I	m	The number of characters to be extracted from the input string.
<return>	DD_STRING	O	m	The specified substring.

B.293 Builtin `strright`

Purpose

The Builtin `strright` extracts letters from the end of a string.

Lexical structure

`strright(string, length)`

The lexical elements of the Builtin `strright` are shown in Table B.293.

Table B.293 – Builtin `strright`

Parameter name	Type	Direction	Usage	Description
string	DD_STRING	I	m	The input string.
length	int	I	m	The number of characters to be extracted from the input string.
<return>	DD_STRING	O	m	The specified substring.

B.294 Builtin `strstr`

Purpose

The Builtin `strstr` returns a sub-string of a string.

Lexical structure

`strstr(string, substring)`

The lexical elements of the Builtin `strstr` are shown in Table B.294.

Table B.294 – Builtin strstr

Parameter name	Type	Direction	Usage	Description
string	DD_STRING	I	m	The string to be searched.
substring	DD_STRING			The search string.
<return>	DD_STRING	O	m	The first occurrence of s1 in s2, or NULL if the string cannot be found.

B.295 Builtin strtrim**Purpose**

The Builtin strtrim trims whitespace from a string.

Lexical structure

strtrim(string)

The lexical elements of the Builtin strtrim are shown in Table B.295.

Table B.295 – Builtin strtrim

Parameter name	Type	Direction	Usage	Description
string	DD_STRING	I	m	The string to be trimmed.
<return>	DD_STRING	O	m	A string with leading and trailing whitespace removed.

B.296 Builtinstrupr**Purpose**

The Builtinstrupr returns an upper-case version of string.

Lexical structure

strupr(string)

The lexical elements of the Builtinstrupr are shown in Table B.296.

Table B.296 – Builtinstrupr

Parameter name	Type	Direction	Usage	Description
string	DD_STRING	I	m	The string to be converted.
<return>	DD_STRING	O	m	An upper-case version of the string.

B.297 Builtin tan**Purpose**

The Builtin tan returns the tangent of a floating point value.

Lexical structure

tan(x)

The lexical elements of the Builtin tan are shown in Table B.297.

Table B.297 – Builtin tan

Parameter name	Type	Direction	Usage	Description
x	double	I	m	A floating point value.

Parameter name	Type	Direction	Usage	Description
<return>	double	O	m	The tangent.

B.298 Builtin tanh

Purpose

The Builtin tanh returns the hyperbolic tangent of a floating point value.

Lexical structure

`tanh(x)`

The lexical elements of the Builtin tanh are shown in Table B.298.

Table B.298 – Builtin tanh

Parameter name	Type	Direction	Usage	Description
x	double	I	m	A floating point value.
<return>	double	O	m	The hyperbolic tangent.

B.299 Builtin Time_To_Date

Purpose

The Builtin Time_To_Date creates a DATE value from a calendar date and time.

Lexical Structure

`Time_To_Date(t)`

The lexical elements of the Builtin Time_To_Date are shown in Table B.299.

Table B.299 – Builtin Time_To_Date

Parameter name	Type	Direction	Usage	Description
t	time_t	I	m	The time_t value.
<return>	long	O	m	The DATE value.

B.300 Builtin TIME_VALUE_to_Hour

Purpose

The Builtin TIME_VALUE_to_Hour extracts the hour component from a TIME_VALUE.

Lexical structure

`TIME_VALUE_to_Hour(time_value)`

The lexical element of the Builtin TIME_VALUE_to_Hour is shown in Table B.300.

Table B.300 – Builtin TIME_VALUE_to_Hour

Parameter name	Type	Direction	Usage	Description
time_value	unsigned long	I	m	The TIME_VALUE.
<return>	int	O	m	The hour component.

B.301 Builtin TIME_VALUE_to_Minute**Purpose**

The Builtin TIME_VALUE_to_Minute extracts the minute component from a TIME_VALUE.

Lexical structure

```
TIME_VALUE_to_Minute(time_value)
```

The lexical element of the Builtin TIME_VALUE_to_Minute is shown in Table B.301.

Table B.301 – Builtin TIME_VALUE_to_Minute

Parameter name	Type	Direction	Usage	Description
time_value	unsigned long	I	m	The TIME_VALUE.
<return>	int	O	m	The minute component.

B.302 Builtin TIME_VALUE_to_Second**Purpose**

The Builtin TIME_VALUE_to_Second extracts the seconds component from a TIME_VALUE.

Lexical structure

```
TIME_VALUE_to_Second(time_value)
```

The lexical element of the Builtin TIME_VALUE_to_Second is shown in Table B.302.

Table B.302 – Builtin TIME_VALUE_to_Second

Parameter name	Type	Direction	Usage	Description
time_value	unsigned long	I	M	The TIME_VALUE.
<return>	int	O	m	The seconds component.

B.303 Builtin TIME_VALUE_to_seconds**Purpose**

The Builtin TIME_VALUE_to_seconds converts a TIME_VALUE to a number of seconds.

Lexical structure

```
TIME_VALUE_to_seconds(time_value)
```

The lexical element of the Builtin TIME_VALUE_to_seconds is shown in Table B.303.

Table B.303 – Builtin TIME_VALUE_to_seconds

Parameter name	Type	Direction	Usage	Description
time_value	unsigned long	I	m	The TIME_VALUE.
<return>	double	O	m	The number of seconds.

B.304 Builtin TIME_VALUE_to_string**Purpose**

The Builtin TIME_VALUE_to_string creates a string representation of a TIME_VALUE. The ANSI C strftime function defines the structure of the format string.

Lexical structure

```
TIME_VALUE_to_string(string, format, time_value)
```

The lexical element of the Builtin TIME_VALUE_to_string is shown in Table B.304.

Table B.304 – Builtin TIME_VALUE_to_string

Parameter name	Type	Direction	Usage	Description
string	DD_STRING	I	m	The string into which formatted TIME_VALUE is copied.
format	DD_STRING	I	m	How the TIME_VALUE is to be formatted.
time_value	unsigned long	I	m	The TIME_VALUE to be formatted.
<return>	int	O	m	The number of character actually copied or -1 is an error occurs.

B.305 Builtin timet_to_string**Purpose**

The Builtin timet_to_string creates a string representation of a time_t. The ANSI C strftime function defines the structure of the format string.

Lexical structure

```
timet_to_string(string, format, timet)
```

The lexical element of the Builtin timet_to_string is shown in Table B.305.

Table B.305 – Builtin timet_to_string

Parameter name	Type	Direction	Usage	Description
string	DD_STRING	I	m	The string into which formatted time_t is copied.
format	DD_STRING	I	m	How the time_t is to be formatted.
timet	time_t	I	m	The time_t to be formatted.
<return>	int	O	m	The number of character actually copied or -1 is an error occurs.

B.306 Builtin timet_to_TIME_VALUE**Purpose**

The Builtin timet_to_TIME_VALUE converts a time_t to a TIME_VALUE.

Lexical structure

```
timet_to_TIME_VALUE(timet)
```

The lexical element of the Builtin timet_to_TIME_VALUE is shown in Table B.306.

Table B.306 – Builtin timet_to_TIME_VALUE

Parameter name	Type	Direction	Usage	Description
timet	time_t	I	m	The time_t value.
<return>	unsigned long	O	m	The TIME value.

B.307 Builtin To_Date

Purpose

The Builtin To_Date creates a DATE from year, month and day.

Lexical Structure

To_Date(year, month, day)

The lexical elements of the Builtin To_Date are shown in Table B.307.

Table B.307 – Builtin To_Date

Parameter name	Type	Direction	Usage	Description
year	int	I	m	The year.
month	int	I	m	The month.
day	int	I	m	The day.
<return>	long	O	m	The DATE value.

B.308 Builtin To_Date_and_Time

Purpose

The Builtin To_Date_and_Time calculates the number of days and fractional days from a given date since 1 January 1900.

NOTE The return value can be assigned to a DATE_AND_TIME variable or can be compared with it.

Lexical structure

To_Date_and_Time(days, hour, minute, second, millisecond)

The lexical elements of the Builtin To_Date_and_Time are shown in Table B.308.

Table B.308 – Builtin To_Date_and_Time

Parameter name	Type	Direction	Usage	Description
days	int	I	m	The number of days.
hour	int	I	m	The hours.
minute	int	I	m	The minutes.
second	int	I	m	The seconds.
millisecond	int	I	m	The milliseconds.
<return>	double	O	m	The number of days and fractional days. The EDD application shall support an implicit typecast between double float and DATE_AND_TIME.

B.309 Builtin To_Time

Purpose

The Builtin To_Time creates a time_t value from its input arguments.

Lexical structure

To_Time(date, hour, minute, second, isDST)

The lexical element of the Builtin To_Time is shown in Table B.309.

Table B.309 – Builtin To_Time

Parameter name	Type	Direction	Usage	Description
date	long	I	m	The date.
hour	int	I	m	The hour.
minute	int	I	m	The minute.
second	int	I	m	The second.
isDST	int	I	m	Specifies whether daylight savings time is in effect. If zero, daylight savings time is not in effect; otherwise, daylight savings time is in effect.
<return>	time_t	O	m	The time_t value.

B.310 Builtin To_TIME_VALUE**Purpose**

The Builtin To_TIME_VALUE creates a TIME_VALUE from its input arguments.

Lexical structure

To_TIME_VALUE(hours, minutes, seconds)

The lexical element of the Builtin To_TIME_VALUE is shown in Table B.310.

Table B.310 – Builtin To_TIME_VALUE

Parameter name	Type	Direction	Usage	Description
hours	int	I	m	The number of hours.
minutes	int	I	m	The number of minutes.
seconds	int	I	m	The number of seconds.
<return>	double	O	m	The TIME_VALUE.

B.311 Builtin trunc**Purpose**

The Builtin trunc truncates a floating-point value.

Lexical structure

trunc(x)

The lexical elements of the Builtin trunc are shown in Table B.311.

Table B.311 – Builtin trunc

Parameter name	Type	Direction	Usage	Description
x	double	I	m	A floating-point value.
<return>	double	O	m	The truncated value.

B.312 Builtin VARID**Purpose**

The Builtin VARID will return the numeric identifier for the variable specified. A valid variable name shall be provided. Each variable in the device description is assigned a unique numeric identifier. This function is to be used when the identifier of a variable either needs to be stored in a temporary buffer, or needs to be sent as a parameter to another Builtin.

Lexical structure

VARID(variable_name)

The lexical elements of the Builtin VARID are shown in Table B.312

Table B.312 – Builtin VARID

Parameter name	Type	Direction	Usage	Description
variable_name	reference	I	m	The identifier of the VARIABLE.
<return>	int	O	m	The variable identifier.

B.313 Builtin vassign

Purpose

The Builtin vassign will assign the value of the source variable to the destination variable. Both variables shall be valid and they shall have the same type.

Lexical structure

vassign(dest_var, source_var)

The lexical elements of the Builtin vassign are shown in Table B.313.

Table B.313 – Builtin vassign

Parameter name	Type	Direction	Usage	Description
dest_var	reference	I	m	The identifier of the destination VARIABLE.
source_var	reference	I	m	The identifier of the source VARIABLE.
<return>	int	O	m	One of the return codes specified in Table B.347.

B.314 Builtin WriteCommand

Purpose

The Builtin WriteCommand allows to write variables with a given command.

Lexical structure

WriteCommand(command_identifier)

The lexical elements of the Builtin WriteCommand are shown in Table B.314.

Table B.314 – Builtin WriteCommand

Parameter name	Type	Direction	Usage	Description
command_identifier	reference	I	m	The Identifier of the COMMAND to execute.
<return>	long	O	m	A protocol specific error value, the encoding of which is defined by the specific protocol or device.

B.315 Builtin XMTR_ABORT_ON_ALL_COMM_STATUS

Purpose

The Builtin XMTR_ABORT_ON_ALL_COMM_STATUS will set all of the bits in the command 48 comm status abort mask. This will cause the system to abort the current method if the device returns any command 48 comm status value. Comm status is defined to be the first octet returned in a transaction, when bit 7 of this octet is set.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function

for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_ABORT_ON_ALL_COMM_STATUS (VOID)

The lexical element of the Builtin XMTR_ABORT_ON_ALL_COMM_STATUS is shown in Table B.315.

Table B.315 – Builtin XMTR_ABORT_ON_ALL_COMM_STATUS

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.316 Builtin XMTR_ABORT_ON_ALL_DATA

Purpose

The Builtin XMTR_ABORT_ON_ALL_DATA will set all of the bits in the command 48 abort masks. This will cause the system to abort if any bit in command 48 is set.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will be valid only during the current method. See send_command for implementation of the masks. See ABORT_ON_RESPONSE_CODE for a list of the available masks, and their default values.

The mask affected by this function is used by the send, send_trans, ext_send, ext_send_trans, get_more_status, and display functions.

Lexical structure

XMTR_ABORT_ON_ALL_DATA (VOID)

The lexical elements of the Builtin XMTR_ABORT_ON_ALL_DATA are shown in Table B.316.

Table B.316 – Builtin XMTR_ABORT_ON_ALL_DATA

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.317 Builtin XMTR_ABORT_ON_ALL_DEVICE_STATUS

Purpose

The Builtin XMTR_ABORT_ON_ALL_DEVICE_STATUS will set all of the bits in the command 48 device status abort mask. This will cause the system to abort the current method if the device returns any command 48 device status value. Device status is defined to be the second data octet returned in a transaction.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_ABORT_ON_ALL_DEVICE_STATUS (VOID)

The lexical element of the Builtin XMTR_ABORT_ON_ALL_DEVICE_STATUS is shown in Table B.317.

Table B.317 – Builtin XMTR_ABORT_ON_ALL_DEVICE_STATUS

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.318 Builtin XMTR_ABORT_ON_ALL_RESPONSE_CODES

Purpose

The Builtin XMTR_ABORT_ON_ALL_RESPONSE_CODES will set all of the bits in the command 48 response code abort mask. This will cause the system to abort the current method if the device returns any command 48 response code value.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_ABORT_ON_ALL_RESPONSE_CODES (VOID)

The lexical element of the Builtin XMTR_ABORT_ON_ALL_RESPONSE_CODES is shown in Table B.318.

Table B.318 – Builtin XMTR_ABORT_ON_ALL_RESPONSE_CODES

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.319 Builtin XMTR_ABORT_ON_COMM_ERROR

Purpose

The Builtin XMTR_ABORT_ON_COMM_ERROR will set the command 48 no comm error mask such that the method will be aborted if a comm error is found while sending command 48.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_ABORT_ON_COMM_ERROR (VOID)

The lexical element of the Builtin XMTR_ABORT_ON_COMM_ERROR is shown in Table B.319.

Table B.319 – Builtin XMTR_ABORT_ON_COMM_ERROR

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.320 Builtin XMTR_ABORT_ON_COMM_STATUS**Purpose**

The Builtin XMTR_ABORT_ON_COMM_STATUS will set the correct bit(s) in the command 48 comm status abort mask such that the specified command 48 comm status value will cause the method to abort. Comm status is defined to be the first data octet returned in a transaction, when bit 7 of this octet is 1. The comm_status_value can be set to a value from 0 to 127, where each bit corresponds to a communications error received from.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_ABORT_ON_COMM_STATUS(comm_status)

The lexical elements of the Builtin XMTR_ABORT_ON_COMM_STATUS are shown in Table B.320.

Table B.320 – Builtin XMTR_ABORT_ON_COMM_STATUS

Parameter name	Type	Direction	Usage	Description
comm_status	int	i	m	The new comm status abort mask.
<return>	VOID	—	—	—

B.321 Builtin XMTR_ABORT_ON_DATA**Purpose**

The Builtin XMTR_ABORT_ON_DATA will set the correct bit(s) in the abort data mask such that the method will be aborted if the specified bit is set in the data field returned by command 48.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_ABORT_ON_DATA(data)

The lexical elements of the Builtin XMTR_ABORT_ON_DATA are shown in Table B.321.

Table B.321 – Builtin XMTR_ABORT_ON_DATA

Parameter name	Type	Direction	Usage	Description
data	int	l	m	The new data abort mask.
<return>	VOID	—	—	—

B.322 Builtin XMTR_ABORT_ON_DEVICE_STATUS**Purpose**

The Builtin XMTR_ABORT_ON_DEVICE_STATUS will set the correct bit(s) in the command 48 device status abort mask in such a way that the specified command 48 device status value will cause the method to abort. Device status is defined to be the second data octet returned in a transaction. The device_status value can be set to a value from 0 to 127, where each bit corresponds to a device status returned from the device.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

Lexical structure

XMTR_ABORT_ON_DEVICE_STATUS(device_status)

The lexical elements of the Builtin XMTR_ABORT_ON_DEVICE_STATUS are shown in Table B.322.

Table B.322 – Builtin XMTR_ABORT_ON_DEVICE_STATUS

Parameter name	Type	Direction	Usage	Description
device_status	int	l	m	The new device status abort mask.
<return>	VOID	—	—	—

B.323 Builtin XMTR_ABORT_ON_NO_DEVICE**Purpose**

The Builtin XMTR_ABORT_ON_NO_DEVICE will set the command 48 no device mask such that the method will be aborted if no device is found while sending a command 48.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_ABORT_ON_NO_DEVICE(VOID)

The lexical element of the Builtin XMTR_ABORT_ON_NO_DEVICE is shown in Table B.323.

Table B.323 – Builtin XMTR_ABORT_ON_NO_DEVICE

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.324 Builtin XMTR_ABORT_ON_RESPONSE_CODE

Purpose

The Builtin XMTR_ABORT_ON_RESPONSE_CODE will set the correct bit(s) in the response code abort mask such that the specified command 48 response code value will cause the method to abort. The response code is defined to be the first data octet returned in a transaction, when bit 7 of this octet is 0.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_ABORT_ON_RESPONSE_CODE(response_code)

The lexical elements of the Builtin XMTR_ABORT_ON_RESPONSE_CODE are shown in Table B.324.

Table B.324 – Builtin XMTR_ABORT_ON_RESPONSE_CODE

Parameter name	Type	Direction	Usage	Description
response_code	int	I	m	The new response code abort mask.
<return>	VOID	—	—	

B.325 Builtin XMTR_IGNORE_ALL_COMM_STATUS

Purpose

The Builtin XMTR_IGNORE_ALL_COMM_STATUS will clear all of the bits in the command 48 comm status retry and abort masks. This will cause the system to ignore all bits in the comm status value. Comm status is defined to be the first data octet returned in a transaction, when bit 7 of this octet is set.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_IGNORE_ALL_COMM_STATUS(VOID)

The lexical element of the Builtin XMTR_IGNORE_ALL_COMM_STATUS is shown in Table B.325.

Table B.325 – Builtin XMTR_IGNORE_ALL_COMM_STATUS

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	

B.326 Builtin XMTR_IGNORE_ALL_DATA

Purpose

The Builtin XMTR_IGNORE_ALL_DATA will clear all of the bits in the command 48 device status retry and abort masks. This will cause the system to ignore all data in the command 48 response message.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will be valid only during the current method. See send_command for implementation of the masks. See ABORT_ON_RESPONSE_CODE for a list of the available masks, and their default values.

The mask affected by this function is used by the send, send_trans, ext_send, ext_send_trans, get_more_status, and display functions.

Lexical structure

XMTR_IGNORE_ALL_DATA (VOID)

The lexical elements of the Builtin XMTR_IGNORE_ALL_DATA are shown in Table B.326.

Table B.326 – Builtin XMTR_IGNORE_ALL_DATA

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.327 Builtin XMTR_IGNORE_ALL_DEVICE_STATUS

Purpose

The Builtin XMTR_IGNORE_ALL_DEVICE_STATUS will clear all of the bits in the command 48 device status retry and abort masks. This will cause the system to ignore all bits in the command 48 device status octet. Device status is defined to be the second data octet returned in a transaction.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_IGNORE_ALL_DEVICE_STATUS (VOID)

The lexical element of the Builtin XMTR_IGNORE_ALL_DEVICE_STATUS is shown in Table B.327.

Table B.327 – Builtin XMTR_IGNORE_ALL_DEVICE_STATUS

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.328 Builtin XMTR_IGNORE_ALL_RESPONSE_CODES

Purpose

The Builtin XMTR_IGNORE_ALL_RESPONSE_CODES will clear all of the bits in the command 48 response code retry and abort masks. This will cause the system to ignore all response code values returned from the device.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the `send_command` function for implementation of the masks. See Builtin `ABORT_ON_RESPONSE_CODE` for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins `send_command`, `send_command_trans`, `ext_send_command`, `ext_send_command_trans`, `get_more_status`, and `display`.

Lexical structure

`XMTR_IGNORE_ALL_RESPONSE_CODES (VOID)`

The lexical element of the Builtin `XMTR_IGNORE_ALL_RESPONSE_CODES` is shown in Table B.328.

Table B.328 – Builtin `XMTR_IGNORE_ALL_RESPONSE_CODES`

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.329 Builtin `XMTR_IGNORE_COMM_ERROR`

Purpose

The Builtin `XMTR_IGNORE_COMM_ERROR` will set the command 48 comm error mask such that a communications error condition will be ignored while sending command 48.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the `send_command` function for implementation of the masks. See `ABORT_ON_RESPONSE_CODE` for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins `send_command`, `send_command_trans`, `ext_send_command`, `ext_send_command_trans`, `get_more_status`, and `display`.

Lexical structure

`XMTR_IGNORE_COMM_ERROR (VOID)`

The lexical element of the Builtin `XMTR_IGNORE_COMM_ERROR` is shown in Table B.329.

Table B.329 – Builtin `XMTR_IGNORE_COMM_ERROR`

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.330 Builtin `XMTR_IGNORE_COMM_STATUS`

Purpose

The Builtin `XMTR_IGNORE_COMM_STATUS` will clear the correct bit(s) in the command 48 comm status abort and retry mask such that the specified bits in the command 48 comm status value will be ignored. Comm status is defined to be first data octet returned in a transaction, when bit 7 of this octet is 1.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the `send_command` for implementation of the masks. See Builtin `ABORT_ON_RESPONSE_CODES` for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins `send_command`, `send_command_trans`, `ext_send_command`, `ext_send_command_trans`, `get_more_status`, and `display`.

Lexical structure

`XMTR_IGNORE_COMM_STATUS(comm_status)`

The lexical elements of the Builtin `XMTR_IGNORE_COMM_STATUS` are shown in Table B.330.

Table B.330 – Builtin `XMTR_IGNORE_COMM_STATUS`

Parameter name	Type	Direction	Usage	Description
<code>comm_status</code>	int	l	m	The new comm status abort and retry mask.
<return>	VOID	—	—	—

B.331 Builtin `XMTR_IGNORE_DEVICE_STATUS`

Purpose

The Builtin `XMTR_IGNORE_DEVICE_STATUS` will clear the correct bit(s) in the command 48 device status abort and retry masks such that the specified bits in the command 48 device status octet are ignored. Device status is defined to be the second data octet returned in a transaction.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the `send_command` function for implementation of the masks. See `ABORT_ON_RESPONSE_CODES` for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins `send_command`, `send_command_trans`, `ext_send_command`, `ext_send_command_trans`, `get_more_status`, and `display`.

Lexical structure

`XMTR_IGNORE_DEVICE_STATUS(device_status)`

The lexical elements of the Builtin `XMTR_IGNORE_DEVICE_STATUS` are shown in Table B.331.

Table B.331 – Builtin `XMTR_IGNORE_DEVICE_STATUS`

Parameter name	Type	Direction	Usage	Description
<code>device_status</code>	int	l	m	The new device status abort and retry mask.
<return>	VOID	—	—	—

B.332 Builtin `XMTR_IGNORE_NO_DEVICE`

Purpose

The Builtin `XMTR_IGNORE_NO_DEVICE` will set the command 48 no device mask to show that the no device condition should be ignored while sending a command 48.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the `send_command` function for implementation of the masks. See `ABORT_ON_RESPONSE_CODES` for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins `send_command`, `send_command_trans`, `ext_send_command`, `ext_send_command_trans`, `get_more_status`, and `display`.

Lexical structure

`XMTR_IGNORE_NO_DEVICE(VOID)`

The lexical element of the Builtin `XMTR_IGNORE_NO_DEVICE` is shown in Table B.332.

Table B.332 – Builtin XMTR_IGNORE_NO_DEVICE

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.333 Builtin XMTR_IGNORE_RESPONSE_CODE

Purpose

The Builtin `XMTR_IGNORE_RESPONSE_CODE` will clear the correct bit(s) in the command 48 response code masks in such a way that the specified command 48 response code value will be ignored. The response code is defined to be the first data octet returned in a transaction, when bit 7 of this octet is 0.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the `send_command` function for implementation of the masks. See `ABORT_ON_RESPONSE_CODES` for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins `send_command`, `send_command_trans`, `ext_send_command`, `ext_send_command_trans`, `get_more_status`, and `display`.

Lexical structure

`XMTR_IGNORE_RESPONSE_CODE(response_code)`

The lexical elements of the Builtin `XMTR_IGNORE_RESPONSE_CODE` are shown in Table B.333.

Table B.333 – Builtin XMTR_IGNORE_RESPONSE_CODE

Parameter name	Type	Direction	Usage	Description
response_code	int	l	m	The new response code mask.
<return>	VOID	—	—	—

B.334 Builtin XMTR_RETRY_ON_ALL_COMM_STATUS

Purpose

The Builtin `XMTR_RETRY_ON_ALL_COMM_STATUS` will clear all of the bits in the command 48 communications status retry and abort masks. This will cause the system to ignore all bits in the communications status value. Communications status is defined to be the first data byte returned in a HART transaction, when bit 7 of this byte is set.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will be valid only during the current method. See `send_command` for implementation of the masks. See `ABORT_ON_RESPONSE_CODE` for a list of the available masks, and their default values.

The mask affected by this function is used by the `send`, `send_trans`, `ext_send`, `ext_send_trans`, `get_more_status`, and `display` functions.

Lexical structure

XMTR_RETRY_ON_ALL_COMM_STATUS (VOID)

The lexical elements of the Builtin XMTR_RETRY_ON_ALL_COMM_STATUS are shown in Table B.334.

Table B.334 – Builtin XMTR_RETRY_ON_ALL_COMM_STATUS

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.335 Builtin XMTR_RETRY_ON_ALL_DATA

Purpose

The Builtin XMTR_RETRY_ON_ALL_DATA will set all of the bits in the command 48 device status retry mask. This will cause the system to retry if any bit in command 48 is set.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will be valid only during the current method. See send_command for implementation of the masks. See ABORT_ON_RESPONSE_CODE for a list of the available masks, and their default values.

The mask affected by this function is used by the send, send_trans, ext_send, ext_send_trans, get_more_status, and display functions.

Lexical structure

XMTR_RETRY_ON_ALL_DATA (VOID)

The lexical elements of the Builtin XMTR_RETRY_ON_ALL_DATA are shown in Table B.335.

Table B.335 – Builtin XMTR_RETRY_ON_ALL_DATA

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.336 Builtin XMTR_RETRY_ON_ALL_DEVICE_STATUS

Purpose

The Builtin XMTR_RETRY_ON_ALL_DEVICE_STATUS will set all of the bits in the command 48 device status retry mask. This will cause the system to retry command 48 if the device returns any command 48 device status value. Device status is defined to be the second data octet returned in a transaction.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODES for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_RETRY_ON_ALL_DEVICE_STATUS (VOID)

The lexical element of the Builtin XMTR_RETRY_ON_ALL_DEVICE_STATUS is shown in Table B.336.

Table B.336 – Builtin XMTR_RETRY_ON_ALL_DEVICE_STATUS

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.337 Builtin XMTR_RETRY_ON_ALL_RESPONSE_CODE**Purpose**

The Builtin XMTR_RETRY_ON_ALL_RESPONSE_CODE will set all of the bits in the command 48 comm status retry mask. This will cause the system to retry command 48 if the device returns any command 48 comm status value. Comm status is defined to be the first data octet returned in a transaction, when bit 7 of this octet is set.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODES for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_RETRY_ON_ALL_RESPONSE_CODE (VOID)

The lexical element of the Builtin XMTR_RETRY_ON_ALL_RESPONSE_CODE is shown in Table B.337.

Table B.337 – Builtin XMTR_RETRY_ON_ALL_RESPONSE_CODE

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.338 Builtin XMTR_RETRY_ON_ALL_RESPONSE_CODES**Purpose**

The Builtin XMTR_RETRY_ON_ALL_RESPONSE_CODE will set all of the bits in the command 48 response code retry mask. This will cause the system command 48 to be retried if the device returns any command 48 response code value.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODES for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_RETRY_ON_ALL_RESPONSE_CODES (VOID)

The lexical element of the Builtin XMTR_RETRY_ON_ALL_RESPONSE_CODES is shown in Table B.338.

Table B.338 – Builtin XMTR_RETRY_ON_ALL_RESPONSE_CODES

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.339 Builtin XMTR_RETRY_ON_COMM_ERROR**Purpose**

The Builtin XMTR_RETRY_ON_COMM_ERROR will set the command 48 no comm error mask such that command 48 will be retried if a comm error is found while sending command 48.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODES for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_RETRY_ON_COMM_ERROR(VOID)

The lexical element of the Builtin XMTR_RETRY_ON_COMM_ERROR is shown in Table B.339.

Table B.339 – Builtin XMTR_RETRY_ON_COMM_ERROR

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.340 Builtin XMTR_RETRY_ON_COMM_STATUS**Purpose**

The Builtin XMTR_RETRY_ON_COMM_STATUS will set the correct bit(s) in the command 48 comm status retry mask in such a way that the specified command 48 comm status value will cause command 48 to be retried. Comm status is defined to be the first data octet returned in a transaction, when bit 7 of this is 1.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODES for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_RETRY_ON_COMM_STATUS(comm_status)

The lexical elements of the Builtin XMTR_RETRY_ON_COMM_STATUS are shown in Table B.340.

Table B.340 – Builtin XMTR_RETRY_ON_COMM_STATUS

Parameter name	Type	Direction	Usage	Description
comm_status	int	l	m	The new comm status retry mask.

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.341 Builtin XMTR_RETRY_ON_DATA

Purpose

The Builtin XMTR_RETRY_ON_DATA will set the correct bit(s) in the retry data mask in such a way that the command 48 will be retried if the specified bit is set in the data field returned by command 48.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODES for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_RETRY_ON_DATA(data)

The lexical elements of the Builtin XMTR_RETRY_ON_DATA are shown in Table B.341.

Table B.341 – Builtin XMTR_RETRY_ON_DATA

Parameter name	Type	Direction	Usage	Description
data	int	l	m	The new data retry mask.
<return>	VOID	—	—	—

B.342 Builtin XMTR_RETRY_ON_DEVICE_STATUS

Purpose

The Builtin XMTR_RETRY_ON_DEVICE_STATUS will set the correct bit(s) in the command 48 device status retry mask in such a way that the specified command 48 device status value will cause command 48 to be retried. Device status is defined to be the second data octet returned in a transaction.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODES for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_RETRY_ON_DEVICE_STATUS(device_status)

The lexical elements of the Builtin XMTR_RETRY_ON_DEVICE_STATUS are shown in Table B.342.

Table B.342 – Builtin XMTR_RETRY_ON_DEVICE_STATUS

Parameter name	Type	Direction	Usage	Description
device_status	INT4	l	m	The new device status retry mask.
<return>	VOID	—	—	—

B.343 Builtin XMTR_RETRY_ON_NO_DEVICE

Purpose

The Builtin XMTR_RETRY_ON_NO_DEVICE will set the command 48 no-device mask in such a way that command 48 will be retried if no device is found while sending command 48.

The retry and abort masks are reset to their values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODES for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_RETRY_ON_NO_DEVICE (VOID)

The lexical element of the Builtin XMTR_RETRY_ON_NO_DEVICE is shown in Table B.343.

Table B.343 – Builtin XMTR_RETRY_ON_NO_DEVICE

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.344 Builtin XMTR_RETRY_ON_RESPONSE_CODE

Purpose

The Builtin XMTR_RETRY_ON_RESPONSE_CODE will set the correct bit(s) in the response code retry mask in such a way that the specified command 48 response code value will cause command 48 to be retried. The response code is defined to be the first data octet returned in a transaction, when bit 7 of this octet is 0.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODES for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_RETRY_ON_RESPONSE_CODE (response_code)

The lexical element of the Builtin XMTR_RETRY_ON_RESPONSE_CODE is shown in Table B.344.

Table B.344 – Builtin XMTR_RETRY_ON_RESPONSE_CODE

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.345 Builtins return codes

Some Builtins have specific return codes. These return codes are specified in Table B.346 and Table B.346. The description of the Table B.346 and Table B.346 is in Table B.345.

Table B.345 – Contents of the return codes description table

Column	Meaning
Value	specifies a long integer returned from the Builtin
Identifier	specifies a reference to a returned long integer value
Description	specifies a description for the return code

Table B.346 – Return code descriptions

Value	Identifier	Description
0	BLTIN_SUCCESS	Success.
1801	BLTIN_NO_MEMORY	Out of memory.
1802	BLTIN_VAR_NOT_FOUND	Cannot find variable.
1803	BLTIN_BAD_ID	Specified item or member ID does not exist.
1804	BLTIN_NO_DATA_TO_SEND	No value to send to device.
1805	BLTIN_WRONG_DATA_TYPE	Wrong variable type for Builtin.
1806	BLTIN_NO_RESP_CODES	Item does not have response codes.
1807	BLTIN_BAD_METHOD_ID	Invalid method ID.
1808	BLTIN_BUFFER_TOO_SMALL	Supplied buffer for starting string is too small.
1809	BLTIN_CANNOT_READ_VARIABLE	Cannot read the specified variable.
1810	BLTIN_INVALID_PROMPT	Bad prompt string.
1811	BLTIN_NO_LANGUAGE_STRING	No valid string for current language.
1812	BLTIN_DDS_ERROR	A DDS error occurred.
1813	BLTIN_FAIL_RESPONSE_CODE	Response code received from the device caused Builtin to fail.
1814	BLTIN_FAIL_COMM_ERROR	Communication error caused Builtin to fail.
1815	BLTIN_NOT_IMPLEMENTED	Builtin function not yet implemented.
1816	BLTIN_BAD_ITEM_TYPE	Wrong item type for Builtin.
1817	BLTIN_VALUE_NOT_SET	A CLASS LOCAL variable has not been initialized before being used.
1818	BLTIN_BLOCK_NOT_FOUND	Specified block does not exist within the device.

Table B.347 – Return code descriptions

Value	Identifier	Description
0	BI_SUCCESS	The Builtin completed successfully.
-1	BI_ERROR	An error occurred during the execution of the Builtin.
-2	BI_ABORT	The user aborted the task during the execution of the Builtin.
-3	BI_NO_DEVICE	There was no response from the device during the execution of the Builtin.
-4	BI_COMM_ERR	A communications error occurred during the execution of the Builtin.

Annex C (informative)

EDD example

C.1 EDD example of a temperature transmitter

The following EDD example describes a temperature transmitter which is composed out of one Device Block, one Analog Input Function Block and one Temperature Technology Block. The general structure of the EDD file is as follows.

- The start is the EDD identification with the keywords according to 7.2.
- There are the general sections for one Temperature Technology Block, one Analog Input Function Block and one Device Block. Each section starts with the block description (block_b type) which is followed by its parameters.
- Each parameter is described by its variable with attributes and its command describing the communication transactions.
- The menu section of the description describes the structure of the operator interface. The design of the screen is tool specific.

Parameter	Value	Unit	State	Name in DD
Main_MENU				Table_Main_Specialist
>> Menu_Operator				Tab_s_Operator
DEVICE_ID	IEC61804 TEMP TR		initial	DeviceBlock_DEVICE_ID
OUT_Value	0.000		valid	AnalogInputFB_OUT_Value
OUT_Status	good		valid	AnalogInputFB_OUT_Status
OUT_SCALE_UnitsIndex	degree_Celsius		initial	AnalogInputFB_OUT_SCALE_UnitsIndex
>> Menu_Specialist				Tab_s_Specialist
>>> Menu_AnalogInputFunctionBlock				Tab_s_AnalogInputFunctionBlock
OUT_Value	0.000		valid	AnalogInputFB_OUT_Value
OUT_Status	good		valid	AnalogInputFB_OUT_Status
OUT_SCALE_EUat100percent	100.000		initial	AnalogInputFB_OUT_SCALE_EUat100percent
OUT_SCALE_EUat0percent	0.000		initial	AnalogInputFB_OUT_SCALE_EUat0percent
OUT_SCALE_UnitsIndex	degree_Celsius		initial	AnalogInputFB_OUT_SCALE_UnitsIndex
OUT_SCALE_DecimalPoint	0		valid	AnalogInputFB_OUT_SCALE_DecimalPoint
HI_LIM	3.400000e+038		initial	AnalogInputFB_HI_LIM
LO_LIM	-3.400000e+038		initial	AnalogInputFB_LO_LIM
>>> Menu_TemperatureTechnologyBlock				Tab_s_TemperatureTechnologyBlock
PRIMARY_VALUE_Value	0.000		valid	TemperatureTB_PRIMARY_VALUE_Value
PRIMARY_VALUE_Status	good		valid	TemperatureTB_PRIMARY_VALUE_Status
PRIMARY_VALUE_UNIT	degree_Celsius		initial	TemperatureTB_PRIMARY_VALUE_UNIT
SECONDARY_VALUE_1_Value	0.000		valid	TemperatureTB_SECONDARY_VALUE_1_Value
SECONDARY_VALUE_1_Status	good		valid	TemperatureTB_SECONDARY_VALUE_1_Status
SECONDARY_VALUE_2_Value	0.000		valid	TemperatureTB_SECONDARY_VALUE_2_Value
SECONDARY_VALUE_2_Status	good		valid	TemperatureTB_SECONDARY_VALUE_2_Status
LIN_TYPE	Ptxxx		valid	TemperatureTB_LIN_TYPE
SENSOR_CONNECTION	4 wires		valid	TemperatureTB_SENSOR_CONNECTION
>>> Menu_DeviceBlock				Tab_s_DeviceBlock
DEVICE_MAN_ID	0		valid	DeviceBlock_DEVICE_MAN_ID
DEVICE_ID	IEC61804 TEMP TR		initial	DeviceBlock_DEVICE_ID
DEVICE_SER_Num	chdlwelskrgwtb		initial	DeviceBlock_DEVICE_SER_Num

Figure C.1 – Example of an operator screen using EDD

IEC 1545/06

Figure C.1 shows an example how the temperature transmitter can be visualised at an operator screen. At the left side there is the menu structure (Main_MENU, Menu_Operator, Menu_Specialist). The Menu_Operator references the measurement value as well its unit and status. The Menu_Specialist has the submenus Analog_Input_Function_Block, Temperature_Technology_Block and Device_Block. On the right-hand side, there is a table-oriented view of the menu details, i.e. the parameters which are referenced in the relevant menu definition. This representation is based on the EDD example in Clause C.2.

C.2 EDD example

/ * -----


```

* Copyright (C) IEC 61804
* Product: EDD
* Device: IEC 61804 Temperature Transmitter
* Description: Electronic Device Description (EDD)
* Dictionary:
* $Revision: Name
* $Date: 2005-07-17
* -----*/

MANUFACTURER 0,
DEVICE_TYPE 0,
DEVICE_REVISION 1,
DD_REVISION 1

/* -----
* DEFINE
* -----*/

#define DEF_STATUS \
    { 0x00 , [bad] }, \
    { 0x40 , [uncertain] }, \
    { 0x80 , [good] }

/* -----
* Device Block
* -----*/

BLOCK DeviceBlock
{
    TYPE PHYSICAL;
    NUMBER 1;
}

//----DB-10----

VARIABLE DeviceBlock__DEVICE_MAN_ID
{
    LABEL [DEVICE_MAN_ID];
    HELP [DeviceBlock__DEVICE_MAN_ID_Help];
    CLASS CONTAINED;
    TYPE UNSIGNED_INTEGER (2);
    HANDLING READ;
}

COMMAND Read__DeviceBlock__DEVICE_MAN_ID
{
    BLOCK DeviceBlock;
    INDEX 10;
    OPERATION READ;
    TRANSACTION
    {
        REQUEST
        {
        }
        REPLY
        {
            DeviceBlock__DEVICE_MAN_ID
        }
    }
}

//----DB-11----

```

```
VARIABLE DeviceBlock__DEVICE_ID
{
    LABEL [DEVICE_ID];
    HELP [DeviceBlock__DEVICE_ID_Help];
    CLASS CONTAINED;
    TYPE ASCII (16)
    {
        DEFAULT_VALUE "IEC61804 TEMP TR";
    }
    HANDLING READ;
}
```

```
COMMAND Read__DeviceBlock__DEVICE_ID
{
    BLOCK DeviceBlock;
    INDEX 11;
    OPERATION READ;
    TRANSACTION
    {
        REQUEST
        {
        }
        REPLY
        {
            DeviceBlock__DEVICE_ID
        }
    }
}
```

//----DB-12----

```
VARIABLE DeviceBlock__DEVICE_SER_Num
{
    LABEL [DEVICE_SER_Num];
    HELP [DeviceBlock__DEVICE_SER_Num_Help];
    CLASS CONTAINED;
    TYPE ASCII (16)
    {
        DEFAULT_VALUE "chdelskfrgwlwtb";
    }
    HANDLING READ;
}
```

```
COMMAND Read__DeviceBlock__DEVICE_SER_Num
{
    BLOCK DeviceBlock;
    INDEX 12;
    OPERATION READ;
    TRANSACTION
    {
        REQUEST
        {
        }
        REPLY
        {
            DeviceBlock__DEVICE_SER_Num
        }
    }
}
```

```
/* -----
* Analog Input Function Block
* -----*/
```

```

BLOCK AnalogInputFB
{
    TYPE FUNCTION;
    NUMBER 1;
}

//----AIFB-10----

VARIABLE AnalogInputFB__OUT__Value
{
    LABEL [OUT__Value];
    HELP [AnalogInputFB__OUT__Help];
    CLASS CONTAINED;
    TYPE FLOAT;
    HANDLING READ;
}

VARIABLE AnalogInputFB__OUT__Status
{
    LABEL [OUT__Status];
    HELP [AnalogInputFB__OUT__Help];
    CLASS CONTAINED;
    TYPE ENUMERATED (1)
    {
        DEF_STATUS
    }
    HANDLING READ;
}

COMMAND Read__AnalogInputFB__OUT
{
    BLOCK AnalogInputFB;
    INDEX 10;
    OPERATION READ;
    TRANSACTION
    {
        REQUEST
        {
        }
        REPLY
        {
            AnalogInputFB__OUT__Value,
            AnalogInputFB__OUT__Status
        }
    }
}

//----AIFB-12----

VARIABLE AnalogInputFB__OUT_SCALE__EUat100percent
{
    LABEL [OUT_SCALE__EUat100percent];
    HELP [AnalogInputFB__OUT_SCALE__EUat100percent__Help];
    CLASS CONTAINED;
    TYPE FLOAT
    {
        DEFAULT_VALUE 100;
    }
    HANDLING READ & WRITE;
}

VARIABLE AnalogInputFB__OUT_SCALE__EUat0percent

```

```

{
    LABEL [OUT_SCALE__EUat0percent];
    HELP [AnalogInputFB__OUT_SCALE__EUat0percent__Help];
    CLASS CONTAINED;
    TYPE FLOAT
    {
        DEFAULT_VALUE 0;
    }
    HANDLING READ & WRITE;
}

VARIABLE AnalogInputFB__OUT_SCALE__UnitsIndex
{
    LABEL [OUT_SCALE__UnitsIndex];
    HELP [AnalogInputFB__OUT_SCALE__UnitsIndex__Help];
    CLASS CONTAINED;
    TYPE ENUMERATED (2)
    {
        DEFAULT_VALUE 1001;
        { 1000 ,[Kelvin], [Kelvin_help] },
        { 1001 ,[degree_Celsius], [degree_Celsius_help] },
        { 1002 ,[degree_Fahrenheit], [degree_Fahrenheit_help] },
        { 1003 ,[degree_Rankine], [degree_Rankine_help] },
        { 1243 ,[millivolt], [millivolt_help] },
        { 1281 ,[Ohm], [Ohm_help] }
    }
    HANDLING READ & WRITE;
}

VARIABLE AnalogInputFB__OUT_SCALE__DecimalPoint
{
    LABEL [OUT_SCALE__DecimalPoint];
    HELP [AnalogInputFB__OUT_SCALE__DecimalPoint__Help];
    CLASS CONTAINED;
    TYPE INTEGER (1);
    HANDLING READ & WRITE;
}

COMMAND Read__AnalogInputFB__OUT_SCALE
{
    BLOCK AnalogInputFB;
    INDEX 12;
    OPERATION READ;
    TRANSACTION
    {
        REQUEST
        {
        }
        REPLY
        {
            AnalogInputFB__OUT_SCALE__EUat100percent,
            AnalogInputFB__OUT_SCALE__EUat0percent,
            AnalogInputFB__OUT_SCALE__UnitsIndex,
            AnalogInputFB__OUT_SCALE__DecimalPoint
        }
    }
}

COMMAND Write__AnalogInputFB__OUT_SCALE
{
    BLOCK AnalogInputFB;
    INDEX 12;
    OPERATION WRITE;
}

```

```

TRANSACTION
{
    REQUEST
    {
        AnalogInputFB__OUT_SCALE__EUat100percent,
        AnalogInputFB__OUT_SCALE__EUat0percent,
        AnalogInputFB__OUT_SCALE__UnitsIndex,
        AnalogInputFB__OUT_SCALE__DecimalPoint
    }
    REPLY
    {
    }
}

//----AIFB-23----

VARIABLE AnalogInputFB__HI_LIM
{
    LABEL [HI_LIM];
    HELP [AnalogInputFB__HI_LIM__Help];
    CLASS CONTAINED;
    TYPE FLOAT
    {
        DEFAULT_VALUE 3.4E+38;
    }
    HANDLING READ & WRITE;
}

COMMAND Read__AnalogInputFB__HI_LIM
{
    BLOCK AnalogInputFB;
    INDEX 23;
    OPERATION READ;
    TRANSACTION
    {
        REQUEST
        {
        }
        REPLY
        {
            AnalogInputFB__HI_LIM
        }
    }
}

COMMAND Write__AnalogInputFB__HI_LIM
{
    BLOCK AnalogInputFB;
    INDEX 23;
    OPERATION WRITE;
    TRANSACTION
    {
        REQUEST
        {
            AnalogInputFB__HI_LIM
        }
        REPLY
        {
        }
    }
}

```

//----AIFB-25----

VARIABLE AnalogInputFB__LO_LIM

```
{
    LABEL [LO_LIM];
    HELP [AnalogInputFB__LO_LIM__Help];
    CLASS CONTAINED;
    TYPE FLOAT
    {
        DEFAULT_VALUE -3.4E+38;
    }
    HANDLING READ & WRITE;
}
```

COMMAND Read__AnalogInputFB__LO_LIM

```
{
    BLOCK AnalogInputFB;
    INDEX 25;
    OPERATION READ;
    TRANSACTION
    {
        REQUEST
        {
        }
        REPLY
        {
            AnalogInputFB__LO_LIM
        }
    }
}
```

COMMAND Write__AnalogInputFB__LO_LIM

```
{
    BLOCK AnalogInputFB;
    INDEX 25;
    OPERATION WRITE;
    TRANSACTION
    {
        REQUEST
        {
            AnalogInputFB__LO_LIM
        }
        REPLY
        {
        }
    }
}
```

```
/* -----
 * Temperature Technology Block
 * -----*/
```

BLOCK TemperatureTB

```
{
    TYPE TRANSDUCER;
    NUMBER 1;
}
```

//----TTB-8----

VARIABLE TemperatureTB__PRIMARY_VALUE__Value

```
{
    LABEL [PRIMARY_VALUE__Value];
}
```

```

    HELP [TemperatureTB__PRIMARY_VALUE__Help];
    CLASS CONTAINED;
    TYPE FLOAT;
    HANDLING READ;
}

VARIABLE TemperatureTB__PRIMARY_VALUE__Status
{
    LABEL [PRIMARY_VALUE__Status];
    HELP [TemperatureTB__PRIMARY_VALUE__Help];
    CLASS CONTAINED;
    TYPE ENUMERATED (1)
    {
        DEF_STATUS
    }
    HANDLING READ;
}

COMMAND Read__TemperatureTB__PRIMARY_VALUE
{
    BLOCK TemperatureTB;
    INDEX 8;
    OPERATION READ;
    TRANSACTION
    {
        REQUEST
        {
        }
        REPLY
        {
            TemperatureTB__PRIMARY_VALUE__Value,
            TemperatureTB__PRIMARY_VALUE__Status
        }
    }
}

//-----TTB-9-----

VARIABLE TemperatureTB__PRIMARY_VALUE__UNIT
{
    LABEL [PRIMARY_VALUE__UNIT];
    HELP [TemperatureTB__PRIMARY_VALUE__UNIT__Help];
    CLASS CONTAINED;
    TYPE ENUMERATED (2)
    {
        DEFAULT_VALUE 1001;
        { 1000 ,[Kelvin], [Kelvin_help] },
        { 1001 ,[degree_Celsius], [degree_Celsius_help] },
        { 1002 ,[degree_Fahrenheit], [degree_Fahrenheit_help] },
        { 1003 ,[degree_Rankine], [degree_Rankine_help] },
        { 1243 ,[millivolt], [millivolt_help] },
        { 1281 ,[Ohm], [Ohm_help] }
    }
    HANDLING READ & WRITE;
}

COMMAND Read__TemperatureTB__PRIMARY_VALUE__UNIT
{
    BLOCK TemperatureTB;
    INDEX 9;
    OPERATION READ;
    TRANSACTION

```

```

{
    REQUEST
    {
    }
    REPLY
    {
        TemperatureTB__PRIMARY_VALUE_UNIT
    }
}

```

COMMAND Write__TemperatureTB__PRIMARY_VALUE_UNIT

```

{
    BLOCK TemperatureTB;
    INDEX 9;
    OPERATION WRITE;
    TRANSACTION
    {
        REQUEST
        {
            TemperatureTB__PRIMARY_VALUE_UNIT
        }
        REPLY
        {
        }
    }
}

```

//----TTB-10----

VARIABLE TemperatureTB__SECONDARY_VALUE_1__Value

```

{
    LABEL [SECONDARY_VALUE_1__Value];
    HELP [TemperatureTB__SECONDARY_VALUE_1__Help];
    CLASS CONTAINED;
    TYPE FLOAT;
    HANDLING READ;
}

```

VARIABLE TemperatureTB__SECONDARY_VALUE_1__Status

```

{
    LABEL [SECONDARY_VALUE_1__Status];
    HELP [TemperatureTB__SECONDARY_VALUE_1__Help];
    CLASS CONTAINED;
    TYPE ENUMERATED (1)
    {
        DEF_STATUS
    }
    HANDLING READ;
}

```

COMMAND Read__TemperatureTB__SECONDARY_VALUE_1

```

{
    BLOCK TemperatureTB;
    INDEX 10;
    OPERATION READ;
    TRANSACTION
    {
        REQUEST
        {
        }
        REPLY
        {
        }
    }
}

```



```

        TemperatureTB__SECONDARY_VALUE_1__Value,
        TemperatureTB__SECONDARY_VALUE_1__Status
    }
}

//-----TTB-11-----

VARIABLE TemperatureTB__SECONDARY_VALUE_2__Value
{
    LABEL [SECONDARY_VALUE_2__Value];
    HELP [TemperatureTB__SECONDARY_VALUE_2__Help];
    CLASS CONTAINED;
    TYPE FLOAT;
    HANDLING READ;
}

VARIABLE TemperatureTB__SECONDARY_VALUE_2__Status
{
    LABEL [SECONDARY_VALUE_2__Status];
    HELP [TemperatureTB__SECONDARY_VALUE_2__Help];
    CLASS CONTAINED;
    TYPE ENUMERATED (1)
    {
        DEF_STATUS
    }
    HANDLING READ;
}

COMMAND Read__TemperatureTB__SECONDARY_VALUE_2
{
    BLOCK TemperatureTB;
    INDEX 11;
    OPERATION READ;
    TRANSACTION
    {
        REQUEST
        {
        }
        REPLY
        {
            TemperatureTB__SECONDARY_VALUE_2__Value,
            TemperatureTB__SECONDARY_VALUE_2__Status
        }
    }
}

//-----TTB-14-----

VARIABLE TemperatureTB__LIN_TYPE
{
    LABEL [LIN_TYPE];
    HELP [TemperatureTB__LIN_TYPE__Help];
    CLASS CONTAINED;
    TYPE ENUMERATED (1)
    {
        { 0 , [No_linearisation]},
        { 102 , "RTD Pt100 a=0.003850 (IEC 751, DIN 43760, JIS C1604-97, BS1904)"},
        { 105 , "RTD Pt1000 a=0.003850 (IEC 751, DIN 43760, JIS C1604-97, BS1904)"},
        { 123 , "RTD Ni100 a=0.006180 (DIN 43760)"},
    }
}

```

```

        { 128 , "TC Type B, Pt30Rh-Pt6Rh (IEC 584, NIST MN 175, DIN
43710, BS 4937, ANSI MC96.1, JIS C1602, NF C42-321)" },
        { 129 , "TC Type C (W5), W5-W26Rh (ASTM E 988)" },
        { 130 , "TC Type D (W3), W3-W25Rh (ASTM E 988)" },
        { 131 , "TC Type E, Ni10Cr-Cu45Ni (IEC584, NIST MN 175, DIN
43710, BS 4937, ANSI MC96.1, JIS C1602, NF C42-321)" },
        { 133 , "TC Type J, Fe-Cu45Ni (IEC 584, NIST MN 175, DIN 43710,
BS 4937, ANSI MC96.1, JIS C1602, NF C42-321)" },
        { 134 , "TC Type K, Ni10Cr-Ni5 (IEC 584, NIST MN 175, DIN 43710,
BS 4937, ANSI MC96.1, JIS C1602, NF C42-321)" },
        { 135 , "TC Type N, Ni14CrSi-NiSi (IEC 584, NIST MN 175,
DIN 43710, BS 4937, ANSI MC96.1, JIS C1602, NF C42-321)" },
        { 136 , "TC Type R, Pt13Rh-Pt (IEC 584, NIST MN 175, DIN 43710,
BS 4937, ANSI MC96.1, JIS C1602, NF C42-321)" },
        { 137 , "TC Type S, Pt10Rh-Pt (IEC 584, NIST MN 175, DIN 43710,
BS 4937, ANSI MC96.1, JIS C1602, NF C42-321)" },
        { 138 , "TC Type T, Cu-Cu45Ni (IEC 584, NIST MN 175, DIN 43710,
BS 4937, ANSI MC96.1, JIS C1602, NF C42-321)" },
        { 139 , "TC Type L, Fe-CuNi (DIN 43710)" },
        { 140 , "TC Type U, Cu-CuNi (DIN 43710)" },
        { 253 , [Ptxxx] }
    }
    HANDLING READ & WRITE;
}

```

```

COMMAND Read__TemperatureTB__LIN_TYPE
{
    BLOCK TemperatureTB;
    INDEX 14;
    OPERATION READ;
    TRANSACTION
    {
        REQUEST
        {
        }
        REPLY
        {
            TemperatureTB__LIN_TYPE
        }
    }
}

```

```

COMMAND Write__TemperatureTB__LIN_TYPE
{
    BLOCK TemperatureTB;
    INDEX 14;
    OPERATION WRITE;
    TRANSACTION
    {
        REQUEST
        {
            TemperatureTB__LIN_TYPE
        }
        REPLY
        {
        }
    }
}

```

//----TTB-36----

VARIABLE TemperatureTB__SENSOR_CONNECTION

```

{
    LABEL [SENSOR_CONNECTION];
    HELP [TemperatureTB__SENSOR_CONNECTION__Help];
    CLASS CONTAINED;
    TYPE ENUMERATED (1)
    {
        { 0 ,"2 wires" },
        { 1 ,"3 wires" },
        { 2 ,"4 wires" }

    }
    HANDLING READ & WRITE;
}

COMMAND Read__TemperatureTB__SENSOR_CONNECTION
{
    BLOCK TemperatureTB;
    INDEX 36;
    OPERATION READ;
    TRANSACTION
    {
        REQUEST
        {
        }
        REPLY
        {
            TemperatureTB__SENSOR_CONNECTION
        }
    }
}

COMMAND Write__TemperatureTB__SENSOR_CONNECTION
{
    BLOCK TemperatureTB;
    INDEX 36;
    OPERATION WRITE;
    TRANSACTION
    {
        REQUEST
        {
            TemperatureTB__SENSOR_CONNECTION
        }
        REPLY
        {
        }
    }
}

/* -----
*   MENU
*   -----*/

MENU Table_Main_Specialist
{
    LABEL [Main_MENU];
    ITEMS
    {
        Tab_s_Operator,
        Tab_s_Specialist
    }
}

MENU Tab_s_Operator

```

```

{
    LABEL [Menu_Operator];
    ITEMS
    {
        DeviceBlock__DEVICE_ID,
        AnalogInputFB__OUT__Value,
        AnalogInputFB__OUT__Status,
        AnalogInputFB__OUT__SCALE__UnitsIndex
    }
}

MENU Tab_s_Specialist
{
    LABEL [Menu_Specialist];
    ITEMS
    {
        Tab_s_AnalogInputFunctionBlock,
        Tab_s_TemperatureTechnologyBlock,
        Tab_s_DeviceBlock
    }
}

MENU Tab_s_AnalogInputFunctionBlock
{
    LABEL [Menu_AnalogInputFunctionBlock];
    ITEMS
    {
        AnalogInputFB__OUT__Value,
        AnalogInputFB__OUT__Status,
        AnalogInputFB__OUT__SCALE__EUat100percent,
        AnalogInputFB__OUT__SCALE__EUat0percent,
        AnalogInputFB__OUT__SCALE__UnitsIndex,
        AnalogInputFB__OUT__SCALE__DecimalPoint,
        AnalogInputFB__HI__LIM,
        AnalogInputFB__LO__LIM
    }
}

MENU Tab_s_TemperatureTechnologyBlock
{
    LABEL [Menu_TemperatureTechnologyBlock];
    ITEMS
    {
        TemperatureTB__PRIMARY_VALUE__Value,
        TemperatureTB__PRIMARY_VALUE__Status,
        TemperatureTB__PRIMARY_VALUE__UNIT,
        TemperatureTB__SECONDARY_VALUE_1__Value,
        TemperatureTB__SECONDARY_VALUE_1__Status,
        TemperatureTB__SECONDARY_VALUE_2__Value,
        TemperatureTB__SECONDARY_VALUE_2__Status,
        TemperatureTB__LIN__TYPE,
        TemperatureTB__SENSOR_CONNECTION
    }
}

MENU Tab_s_DeviceBlock
{
    LABEL [Menu_DeviceBlock];
    ITEMS
    {
        DeviceBlock__DEVICE_MAN_ID,
        DeviceBlock__DEVICE_ID,
        DeviceBlock__DEVICE_SER_Num
    }
}

```

}
}

IECNORM.COM :: Click to view the full PDF of IEC 61804-3:2010

Annex D (normative)

Profiles of EDDL and Builtins

D.1 Conventions for profiles of EDDL and Builtins

Annex D provides Profiles of EDDL and Builtins. Profiles means to select from this document the appropriate specifications for a specific profile. D.1 provides conventions and contains templates for selection tables. The selection tables define which EDDL constructs and syntax are used and which Builtins are selected by the various consortia.

The following conventions in Table D.1, Table D.2, and Table D.3 for the profiles apply.

Table D.1 – Profile selection tables

Lexical construct	Attribute	Presence	Constraints

Table D.2 – EDDL Formal Definition profile tables

Reference	Title	Presence	Constraints

Table D.3 – Contents of selection tables

Column	Text	Meaning
Lexical construct		is a lexical construct element of the EDDL
Attributes	<text>	is one of the attributes of a lexical construct element
Presence	NO	this (sub)clause is not included in the profile
	YES	this (sub)clause is fully (100 %) included in the profile in this case no further detail is given
Constraints	—	no constraints other than given in the reference document (sub)clause, or not applicable
	<text>	the text defines the constraint directly, for longer text table footnotes or table notes may be used
Reference	<#>	clause or annex number
Title	<text>	title of heading

D.2 Profiles for PROFIBUS and PROFINET

D.2.1 EDDL profile

Table D.4 is the selection of elements from the lexical structure constructs specified in Clause 7 and used by the consortium PROFIBUS&PROFINET International (PI). PROFIBUS⁶ and PROFINET⁶ are specified in IEC 61784-1 CPF3 and IEC 61784-2 CPF3.

⁶ PROFIBUS & PROFINET are trade names of PROFIBUS Nutzerorganisation e.V. (PNO) as part of PROFIBUS & PROFINET International. PNO is a non-profit trade organization to support the fieldbus technologies PROFIBUS & PROFINET. This information is given for the convenience of users of this standard and does not constitute an endorsement by IEC of the trade name holder or any of its products. Compliance to this profile

Table D.4 – EDDL element selection for PROFIBUS&PROFINET

Reference	Basic construct	Attribute	Presence	Constraints
7.2.2.1	DD_REVISION		YES	—
7.2.2.2	DEVICE_REVISION		YES	—
7.2.2.3	DEVICE_TYPE		YES	—
7.2.2.4	EDD_PROFILE		YES	—
7.2.2.5	EDD_VERSION		YES	—
7.2.2.6	MANUFACTURER		YES	—
7.2.2.7	MANUFACTURER_EXT		YES	—
7.3	AXIS		YES	—
7.3.2.1		CONSTANT_UNIT	YES	—
7.36.8		HELP	YES	—
7.36.9		LABEL	YES	—
7.3.2.2		MAX_VALUE	YES	—
7.3.2.3		MIN_VALUE	YES	—
7.3.2.4		SCALING	YES	—
7.4.1	BLOCK_A		NO	—
7.4.2	BLOCK_B		YES	—
7.4.2.2.1		NUMBER	YES	—
7.4.2.2.2		TYPE	YES	—
7.5	CHART		YES	—
7.36.10		MEMBERS	YES	—
7.5.2.1		CYCLE_TIME	YES	—
7.36.7		HEIGHT	YES	—
7.36.8		HELP	YES	—
7.36.9		LABEL	YES	—
7.5.2.2		LENGTH	YES	—
7.5.2.3		TYPE	YES	—
7.36.16		VALIDITY	YES	—
7.36.17		WIDTH	YES	—
7.6	COLLECTION		YES	—
7.6.2		item-type	YES	Except: BLOCK_A, VARIABLE_LIST, PROGRAM, DOMAIN
7.36.10		MEMBERS	YES	—
7.36.8		HELP	YES	—
7.36.9		LABEL	YES	—
7.36.16		VALIDITY	YES	—
7.7	COMMAND		YES	—
7.7.2.1		OPERATION	YES	—
7.7.2.2		TRANSACTION	YES	—
7.7.2.3		INDEX	YES	—
7.7.2.4		BLOCK_B	YES	—
7.7.2.5		NUMBER	NO	—
7.7.2.6		SLOT	YES	—
7.7.2.7		SUB_SLOT	YES	PROFINET devices only.
7.7.2.8		CONNECTION	YES	—
7.7.2.9		HEADER	NO	—
7.7.2.10		MODULE	NO	—
7.36.14		RESPONSE_CODES	YES	—
7.8	COMPONENT		YES	—
7.36.9		LABEL	YES	—
7.8.2.1		CAN_DELETE	YES	—
7.8.2.2		CHECK_CONFIGURATION	YES	—
7.36.1		CLASSIFICATION	YES	—
7.36.2		COMPONENT_PARENT	YES	—
7.36.3		COMPONENT_PATH	YES	—
7.8.2.3		COMPONENT_RELATIONS	YES	—
7.8.2.4		DECLARATION	NO	—
7.8.2.5		DETECT	YES	—
7.8.2.6		EDD	YES	—
7.36.8		HELP	YES	—
7.8.2.7		INITIAL_VALUES	YES	—
7.36.13		PROTOCOL	YES	—
7.8.2.8		REDUNDANCY	YES	—
7.8.2.9		SCAN	YES	—
7.8.2.10		SCAN_LIST	YES	—

does not require use of the tradename. Use of the trade names requires permission from the trade name holder.

Reference	Basic construct	Attribute	Presence	Constraints
7.36.15		SUPPLIED_INTERFACE	NO	—
7.9	COMPONENT_FOLDER		YES	—
7.36.9		LABEL	YES	—
7.36.1		CLASSIFICATION	YES	—
7.36.2		COMPONENT_PARENT	YES	—
7.36.3		COMPONENT_PATH	YES	—
7.36.8		HELP	YES	—
7.36.15		SUPPLIED_INTERFACE	YES	—
7.10	COMPONENT_REFERENCE		YES	—
7.36.1		CLASSIFICATION	YES	—
7.36.2		COMPONENT_PARENT	YES	—
7.36.3		COMPONENT_PATH	YES	—
7.2.2.2		DEVICE_REVISION	YES	—
7.2.2.3		DEVICE_TYPE	YES	—
7.2.2.6		MANUFACTURER	YES	—
7.36.15		SUPPLIED_INTERFACE	YES	—
7.11	COMPONENT_RELATION		YES	—
7.11.2.1		COMPONENTS	YES	—
7.36.9		LABEL	YES	—
7.36.8		HELP	YES	—
7.11.2.2		RELATION_TYPE	YES	—
7.11.2.3		ADDRESSING	YES	—
7.11.2.4		MAXIMUM_NUMBER	YES	—
7.11.2.5		MINIMUM_NUMBER	YES	—
7.11.2.6		REQUIRED_INTERFACE	NO	—
7.36.15		SUPPLIED_INTERFACE	NO	—
7.12	CONNECTION		YES	—
7.12.2		APPINSTANCE	YES	—
7.13	DOMAIN		NO	—
7.14	EDIT_DISPLAY		NO	—
7.15	FILE		YES	—
7.36.12		MEMBERS	YES	—
7.36.8		HELP	YES	—
7.36.9		LABEL	YES	—
7.16	GRAPH		YES	—
7.36.12		MEMBERS	YES	—
7.16.2.1		CYCLE_TIME	YES	—
7.36.7		HEIGHT	YES	—
7.36.8		HELP	YES	—
7.36.9		LABEL	YES	—
7.36.16		VALIDITY	YES	—
7.36.17		WIDTH	YES	—
7.16.2.2		X_AXIS	YES	—
7.17	GRID		YES	—
7.17.2.1		VECTORS	YES	—
7.36.6		HANDLING	YES	—
7.36.7		HEIGHT	YES	—
7.36.8		HELP	YES	—
7.36.9		LABEL	YES	—
7.17.2.2		ORIENTATION	YES	—
7.36.16		VALIDITY	YES	—
7.36.17		WIDTH	YES	—
7.18	IMAGE		YES	—
7.18.2.1		PATH	YES	—
7.36.8		HELP	YES	—
7.36.9		LABEL	YES	—
7.18.2.2		LINK	YES	—
7.36.16		VALIDITY	YES	—
7.19	IMPORT		YES	—
7.20	INTERFACE		NO	—
7.21	LIKE		YES	—
7.22	LIST		YES	—
7.22.2.1		TYPE	YES	—
7.22.2.2		CAPACITY	YES	—
7.22.2.3		COUNT	YES	—
7.36.8		HELP	YES	—
7.36.9		LABEL	YES	—
7.36.16		VALIDITY	YES	—
7.23	MENU		YES	—
7.23.2.1		ITEMS	YES	Except: BLOCK_A parameters,

Reference	Basic construct	Attribute	Presence	Constraints
				elements of BLOCK_A parameters, qualifier DISPLAY_VALUE
7.36.9		LABEL	YES	—
7.23.2.2		ACCESS	YES	—
7.36.8		HELP	YES	—
7.23.2.3		POST_EDIT_ACTIONS	YES	—
7.23.2.4		POST_READ_ACTIONS	YES	—
7.23.2.5		POST_WRITE_ACTIONS	YES	—
7.23.2.6		PRE_EDIT_ACTIONS	YES	—
7.23.2.7		PRE_READ_ACTIONS	YES	—
7.23.2.8		PRE_WRITE_ACTIONS	YES	—
7.23.2.9		PURPOSE	YES	—
7.23.2.10		ROLE	YES	—
7.23.2.11		STYLE	YES	—
7.36.16		VALIDITY	YES	—
7.24	METHOD		YES	—
7.36.1		DEFINITION	YES	—
7.24.2.1		ACCESS	YES	—
7.24.2.2		CLASS	YES	Except: HART
7.36.9		LABEL	YES	—
7.36.8		HELP	YES	—
7.24.2.3		TYPE	YES	—
7.36.16		VALIDITY	YES	—
7.25	PROGRAM		NO	—
7.26	RECORD		NO	—
7.27	REFERENCE_ARRAY		YES	—
7.6.2		item-type	YES	Except: BLOCK_A, VARIABLE_LIST, PROGRAM, DOMAIN
7.27.2		ELEMENTS	YES	—
7.36.8		HELP	YES	—
7.36.9		LABEL	YES	—
7.36.16		VALIDITY	YES	—
7.28.1	REFRESH		YES	—
7.28.2	UNIT		YES	—
7.28.3	WRITE_AS_ONE		YES	—
7.36.14	RESPONSE_CODES		YES	—
7.30	SOURCE		YES	—
7.36.12		MEMBERS	YES	—
7.36.5		EMPHASIS	YES	—
7.30.2.1		EXIT_ACTIONS	YES	—
7.36.8		HELP	YES	—
7.30.2.2		INIT_ACTIONS	YES	—
7.36.9		LABEL	YES	—
7.36.10		LINE_COLOR	YES	—
7.36.11		LINE_TYPE	YES	—
7.30.2.3		REFRESH_ACTIONS	YES	—
7.36.16		VALIDITY	YES	—
7.30.2.4		Y_AXIS	YES	—
7.31	TEMPLATE		YES	—
7.31.2		DEFAULT_VALUES	YES	—
7.36.8		HELP	YES	—
7.36.9		LABEL	YES	—
7.32	VALUE_ARRAY		YES	—
7.36.9		LABEL	YES	—
7.32.2.1		NUMBER_OF_ELEMENTS	YES	—
7.32.2.2		TYPE	YES	—

Reference	Basic construct	Attribute	Presence	Constraints
7.36.8		HELP	YES	—
7.36.14		RESPONSE_CODES	YES	—
7.36.16		VALIDITY	YES	—
7.33	VARIABLE		YES	—
7.33.2.1		CLASS	YES	Except: HART
7.36.9		LABEL	YES	—
7.33.2.2		TYPE	YES	Initial value of INTEGER and UNSIGNED_INTEGER VARIABLES is 1
7.33.2.3		CONSTANT_UNIT	YES	—
7.33.2.4		DEFAULT_VALUE	YES	—
7.36.6		HANDLING	YES	—
7.36.8		HELP	YES	—
7.33.2.5		INITIAL_VALUE	YES	—
7.33.2.6		POST_EDIT_ACTIONS	YES	—
7.33.2.7		POST_READ_ACTIONS	YES	—
7.33.2.8		POST_WRITE_ACTIONS	YES	—
7.33.2.9		PRE_EDIT_ACTIONS	YES	—
7.33.2.10		PRE_READ_ACTIONS	YES	—
7.33.2.11		PRE_WRITE_ACTIONS	YES	—
7.33.2.12		READ_TIMEOUT	YES	—
7.33.2.12		WRITE_TIMEOUT	YES	—
7.33.2.13		REFRESH_ACTIONS	YES	—
7.36.14		RESPONSE_CODE	NO	—
7.33.2.14		STYLE	YES	—
7.36.16		VALIDITY	YES	—
7.34	VARIABLE_LIST		NO	—
7.35	WAVEFORM		YES	—
7.35.2.1		TYPE	YES	—
7.36.5		EMPHASIS	YES	—
7.35.2.2		EXIT_ACTIONS	YES	—
7.36.6		HANDLING	YES	—
7.36.8		HELP	YES	—
7.35.2.3		INIT_ACTIONS	YES	—
7.35.2.4		KEY_POINTS	YES	—
7.36.9		LABEL	YES	—
7.36.10		LINE_COLOR	YES	—
7.36.11		LINE_TYPE	YES	—
7.35.2.5		REFRESH_ACTIONS	YES	—
7.36.16		VALIDITY	YES	—
7.35.2.6		Y_AXIS	YES	—
7.37	Conditional expressions		YES	—
7.38	Referencing		YES	—
7.39	Strings		YES	—
7.40	Expressions		YES	—
7.41	Text dictionary		YES	—
NOTE Text is only ISO Latin 1.				

D.2.2 Builtin profile

Table D.5 is the selection of Builtins specified in Annex B and used by the consortium PROFIBUS&PROFINET International (PI).

The Builtins are listed in alphabetical order.

Table D.5 – Builtin profile for PI

Reference	Builtin Name	Presence	Constraints
B.3	_ERROR	YES	—
B.4	_TRACE	YES	—
B.5	_WARNING	YES	—
B.6	abort	YES	—
B.7	abort_on_all_comm_errors	NO	—
B.8	ABORT_ON_ALL_COMM_STATUS	NO	—
B.9	ABORT_ON_ALL_DEVICE_STATUS	NO	—
B.10	ABORT_ON_ALL_RESPONSE_CODES	NO	—
B.11	abort_on_all_response_codes	NO	—
B.12	ABORT_ON_COMM_ERROR	NO	—
B.13	abort_on_comm_error	NO	—

Reference	Builtin Name	Presence	Constraints
B.14	ABORT_ON_COMM_STATUS	NO	—
B.15	ABORT_ON_DEVICE_STATUS	NO	—
B.16	ABORT_ON_NO_DEVICE	NO	—
B.17	ABORT_ON_RESPONSE_CODE	NO	—
B.18	abort_on_response_code	NO	—
B.19	abs	YES	—
B.20	ACKNOWLEDGE	YES	—
B.21	acknowledge	YES	—
B.22	acos	YES	—
B.23	add_abort_method (version A)	NO	—
B.24	add_abort_method (version B)	YES	—
B.25	AddTime	YES	—
B.26	asin	YES	—
B.27	assign	NO	—
B.28	assign_double	YES	—
B.29	assign_float	YES	—
B.30	assign_int	YES	—
B.31	assign_var	YES	—
B.32	assign2	NO	—
B.33	atan	YES	—
B.34	ATOF	NO	—
B.35	atof	YES	—
B.36	ATOI	NO	—
B.37	atoi	YES	—
B.38	BUILD_MESSAGE	YES	—
B.39	ByteToDouble	YES	—
B.40	ByteToFloat	YES	—
B.41	ByteToLong	YES	—
B.42	ByteToShort	YES	—
B.43	cbrt	YES	—
B.44	ceil	YES	—
B.45	cos	YES	—
B.46	cosh	YES	—
B.47	dassign	YES	—
B.48	DATE_AND_TIME_VALUE_to_string	YES	—
B.49	Date_to_DayOfMonth	YES	—
B.50	DATE_to_days	YES	—
B.51	Date_to_Month	YES	—
B.52	DATE_to_string	YES	—
B.53	Date_To_Time	YES	—
B.54	Date_to_Year	YES	—
B.55	days_to_DATE	YES	—
B.56	DELAY	YES	—
B.57	delay	YES	—
B.58	DELAY_TIME	YES	—
B.59	delayfor	NO	—
B.60	delayfor2	NO	—
B.61	DICT_ID	NO	—
B.62	dictionary_string	YES	—
B.63	DiffTime	YES	—
B.64	discard_on_exit	YES	—
B.65	DISPLAY	YES	—
B.66	display	YES	—
B.67	display_bitenum	YES	—
B.68	display_builtin_error	NO	—
B.69	display_comm_error	NO	—
B.70	display_comm_status	NO	—
B.71	display_device_status	NO	—
B.72	display_dynamics	NO	—
B.73	display_dynamics2	NO	—
B.74	display_message	NO	—
B.75	display_message2	NO	—
B.76	display_response_code	NO	—
B.77	display_response_status	NO	—
B.78	display_xmtr_status	NO	—
B.79	DoubleToByte	YES	—
B.80	edit_device_value	NO	—
B.81	edit_device_value2	NO	—
B.82	edit_local_value	NO	—
B.83	edit_local_value2	NO	—
B.84	exp	YES	—
B.85	ext_send_command	NO	—

Reference	Builtin Name	Presence	Constraints
B.86	ext_send_command_trans	NO	—
B.87	fail_on_all_comm_errors	NO	—
B.88	fail_on_all_response_codes	NO	—
B.89	fail_on_comm_error	NO	—
B.90	fail_on_response_code	NO	—
B.91	fassign	YES	—
B.92	fgetval	YES	—
B.93	float_value	YES	—
B.94	FloatToByte	YES	—
B.95	floor	YES	—
B.96	fmod	YES	—
B.97	fpclassify	YES	—
B.98	From_DATE_AND_TIME_VALUE	YES	—
B.99	From_TIME_VALUE	YES	—
B.100	fsetval	YES	—
B.101	ftoa	YES	—
B.102	fvar_value	YES	—
B.103	get_acknowledgement	NO	—
B.104	get_acknowledgement2	NO	—
B.105	get_block_instance_by_object_index	NO	—
B.106	get_block_instance_by_tag	NO	—
B.107	get_block_instance_count	NO	—
B.108	get_comm_error	NO	—
B.109	get_comm_error_string	NO	—
B.110	get_date	NO	—
B.111	get_date_lelem	NO	—
B.112	get_date_lelem2	NO	—
B.113	get_date_value	NO	—
B.114	get_date_value2	NO	—
B.115	GET_DD_REVISION	YES	—
B.116	get_dds_error	NO	—
B.117	GET_DEV_VAR_VALUE	YES	—
B.118	get_dev_var_value	YES	—
B.119	GET_DEVICE_REVISION	YES	—
B.120	GET_DEVICE_TYPE	YES	—
B.121	get_dictionary_string	YES	—
B.122	get_double	NO	—
B.123	get_double_lelem	NO	—
B.124	get_double_lelem2	NO	—
B.125	get_double_value	NO	—
B.126	get_double_value2	NO	—
B.127	get_enum_string	YES	—
B.128	get_float	NO	—
B.129	get_float_lelem	NO	—
B.130	get_float_lelem2	NO	—
B.131	get_float_value	NO	—
B.132	get_float_value2	NO	—
B.133	GET_LOCAL_VAR_VALUE	YES	—
B.134	get_local_var_value	YES	—
B.135	GET_MANUFACTURER	YES	—
B.136	get_more_status	NO	—
B.137	get_resolve_status	NO	—
B.138	get_response_code	NO	—
B.139	get_response_code_string	NO	—
B.140	get_rspcode_string	NO	—
B.141	get_rspcode_string_by_id	YES	—
B.142	get_signed	NO	—
B.143	get_signed_lelem	NO	—
B.144	get_signed_lelem2	NO	—
B.145	get_signed_value	NO	—
B.146	get_signed_value2	NO	—
B.147	get_status_code_string	NO	—
B.148	get_status_string	NO	—
B.149	get_stddict_string	NO	—
B.150	get_string	NO	—
B.151	get_string_lelem	NO	—
B.152	get_string_lelem2	NO	—
B.153	get_string_value	NO	—
B.154	get_string_value2	NO	—
B.155	GET_TICK_COUNT	YES	—
B.156	get_unsigned	NO	—
B.157	get_unsigned_lelem	NO	—

Reference	Builtin Name	Presence	Constraints
B.158	get_unsigned_lelem2	NO	—
B.159	get_unsigned_value	NO	—
B.160	get_unsigned_value2	NO	—
B.161	get_variable_string	YES	—
B.162	GetCurrentDate	YES	—
B.163	GetCurrentDateAndTime	YES	—
B.164	GetCurrentTime	YES	—
B.165	iassign	YES	—
B.166	igetval	YES	—
B.167	IGNORE_ALL_COMM_STATUS	NO	—
B.168	IGNORE_ALL_DEVICE_STATUS	NO	—
B.169	IGNORE_ALL_RESPONSE_CODES	NO	—
B.170	IGNORE_COMM_ERROR	NO	—
B.171	IGNORE_COMM_STATUS	NO	—
B.172	IGNORE_DEVICE_STATUS	NO	—
B.173	IGNORE_NO_DEVICE	YES	—
B.174	IGNORE_RESPONSE_CODE	NO	—
B.175	int_value	YES	—
B.176	is_NaN	YES	—
B.177	isetval	YES	—
B.178	ITEM_ID	NO	—
B.179	ITOA	NO	—
B.180	itoa (version A)	YES	—
B.181	itoa (version B)	NO	—
B.182	ivar_value	YES	—
B.183	lassign	YES	—
B.184	lgetval	NO	—
B.185	ListDeleteElementAt	YES	—
B.186	ListDeleteElementAt2	NO	—
B.187	ListInsert	YES	—
B.188	ListInsert2	NO	—
B.189	log	YES	—
B.190	LOG_MESSAGE	YES	—
B.191	log10	YES	—
B.192	log2	YES	—
B.193	long_value	YES	—
B.194	LongToByte	YES	—
B.195	lsetval	YES	—
B.196	lvar_value	YES	—
B.197	Make_Time	YES	—
B.198	MEMBER_ID	NO	—
B.199	MenuDisplay	YES	—
B.200	method_abort	NO	—
B.201	nan	YES	—
B.202	NaN_value	YES	—
B.203	nanf	YES	—
B.204	ObjectReference	YES	—
B.205	pop_abort_method	YES	—
B.206	pow	YES	—
B.207	process_abort	YES	—
B.208	push_abort_method	YES	—
B.209	put_date	NO	—
B.210	put_date_value	NO	—
B.211	put_date_value2	NO	—
B.212	put_double	NO	—
B.213	put_double_value	NO	—
B.214	put_double_value2	NO	—
B.215	put_float	NO	—
B.216	put_float_value	NO	—
B.217	put_float_value2	NO	—
B.218	PUT_MESSAGE	YES	—
B.219	put_message	YES	—
B.220	put_signed	NO	—
B.221	put_signed_value	NO	—
B.222	put_signed_value2	NO	—
B.223	put_string	NO	—
B.224	put_string_value	NO	—
B.225	put_string_value2	NO	—
B.226	put_unsigned	NO	—
B.227	put_unsigned_value	NO	—
B.228	put_unsigned_value2	NO	—
B.229	read_value	NO	—

Reference	Builtin Name	Presence	Constraints
B.230	read_value2	NO	—
B.231	ReadCommand	YES	—
B.232	remove_abort_method (version A)	YES	—
B.233	remove_abort_method (version B)	NO	—
B.234	remove_all_abort_methods	YES	—
B.235	resolve_array_ref	NO	—
B.236	resolve_block_ref	NO	—
B.237	resolve_list_ref	NO	—
B.238	resolve_local_ref	NO	—
B.239	resolve_local_ref2	NO	—
B.240	resolve_param_list_ref	NO	—
B.241	resolve_param_ref	NO	—
B.242	resolve_param_ref2	NO	—
B.243	resolve_record_ref	NO	—
B.244	resolve_record_ref2	NO	—
B.245	ret_double_value	NO	—
B.246	ret_double_value2	NO	—
B.247	ret_float_value	NO	—
B.248	ret_float_value2	NO	—
B.249	ret_signed_value	NO	—
B.250	ret_signed_value2	NO	—
B.251	ret_unsigned_value	NO	—
B.252	ret_unsigned_value2	NO	—
B.253	retry on all comm errors	NO	—
B.254	RETRY ON ALL COMM STATUS	NO	—
B.255	RETRY ON ALL DEVICE STATUS	NO	—
B.256	RETRY ON ALL RESPONSE_CODES	NO	—
B.257	retry on all response codes	NO	—
B.258	RETRY_ON_COMM_ERROR	NO	—
B.259	retry on comm error	NO	—
B.260	RETRY_ON_COMM_STATUS	NO	—
B.261	RETRY_ON_DEVICE_STATUS	NO	—
B.262	RETRY_ON_NO_DEVICE	NO	—
B.263	RETRY_ON_RESPONSE_CODE	NO	—
B.264	retry on response code	NO	—
B.265	round	YES	—
B.266	save_on_exit	YES	—
B.267	save values	YES	—
B.268	seconds to TIME VALUE	YES	—
B.269	SELECT FROM LIST	YES	—
B.270	select from list	YES	—
B.271	select from menu	NO	—
B.272	select from menu2	NO	—
B.273	send	NO	—
B.274	send all values	NO	—
B.275	send command	NO	—
B.276	send command trans	NO	—
B.277	send on exit	YES	—
B.278	send trans	NO	—
B.279	send_value	NO	—
B.280	send_value2	NO	—
B.281	SET_NUMBER_OF_RETRIES	YES	—
B.282	sgetval	YES	—
B.283	ShortToByte	YES	—
B.284	sin	YES	—
B.285	sinh	YES	—
B.286	sqrt	YES	—
B.287	ssetval	YES	—
B.288	strcmp	YES	—
B.289	strleft	YES	—
B.290	strlen	YES	—
B.291	strlwr	YES	—
B.292	strmid	YES	—
B.293	strright	YES	—
B.294	strstr	YES	—
B.295	strtrim	YES	—
B.296	strupr	YES	—
B.297	tan	YES	—
B.298	tanh	YES	—
B.299	Time To Date	YES	—
B.300	TIME VALUE to Hour	YES	—
B.301	TIME_VALUE to Minute	YES	—

Reference	Builtin Name	Presence	Constraints
B.302	TIME_VALUE_to_Second	YES	—
B.303	TIME_VALUE_to_seconds	YES	—
B.304	TIME_VALUE_to_string	YES	—
B.305	timet_to_string	YES	—
B.306	timet_to_TIME_VALUE	YES	—
B.307	To_Date	YES	—
B.308	To Date and Time	YES	—
B.309	To Time	YES	—
B.310	To_TIME_VALUE	YES	—
B.311	trunc	YES	—
B.312	VARID	YES	—
B.313	vassign	YES	—
B.314	WriteCommand	YES	—
B.315	XMTR_ABORT_ON_ALL_COMM_STATUS	NO	—
B.316	XMTR_ABORT_ON_ALL_DATA	NO	—
B.317	XMTR_ABORT_ON_ALL_DEVICE_STATUS	NO	—
B.318	XMTR_ABORT_ON_ALL_RESPONSE_CODES	NO	—
B.319	XMTR_ABORT_ON_COMM_ERROR	NO	—
B.320	XMTR_ABORT_ON_COMM_STATUS	NO	—
B.321	XMTR_ABORT_ON_DATA	NO	—
B.322	XMTR_ABORT_ON_DEVICE_STATUS	NO	—
B.323	XMTR_ABORT_ON_NO_DEVICE	NO	—
B.324	XMTR_ABORT_ON_RESPONSE_CODE	NO	—
B.325	XMTR_IGNORE_ALL_COMM_STATUS	NO	—
B.326	XMTR_IGNORE_ALL_DATA	NO	—
B.327	XMTR_IGNORE_ALL_DEVICE_STATUS	NO	—
B.328	XMTR_IGNORE_ALL_RESPONSE_CODES	NO	—
B.329	XMTR_IGNORE_COMM_ERROR	NO	—
B.330	XMTR_IGNORE_COMM_STATUS	NO	—
B.331	XMTR_IGNORE_DEVICE_STATUS	NO	—
B.332	XMTR_IGNORE_NO_DEVICE	NO	—
B.333	XMTR_IGNORE_RESPONSE_CODE	NO	—
B.334	XMTR_RETRY_ON_ALL_COMM_STATUS	NO	—
B.335	XMTR_RETRY_ON_ALL_DATA	NO	—
B.336	XMTR_RETRY_ON_ALL_DEVICE_STATUS	NO	—
B.337	XMTR_RETRY_ON_ALL_RESPONSE_CODE	NO	—
B.338	XMTR_RETRY_ON_ALL_RESPONSE_CODES	NO	—
B.339	XMTR_RETRY_ON_COMM_ERROR	NO	—
B.340	XMTR_RETRY_ON_COMM_STATUS	NO	—
B.341	XMTR_RETRY_ON_DATA	NO	—
B.342	XMTR_RETRY_ON_DEVICE_STATUS	NO	—
B.343	XMTR_RETRY_ON_NO_DEVICE	NO	—
B.344	XMTR_RETRY_ON_RESPONSE_CODE	NO	—
B.345	Return Codes	YES	—

D.2.3 EDDL Formal Definition profile

Reference	Title	Presence	Constraints
A.1	EDDL preprocessor	YES	—
A.2	Conventions	YES	—
A.3	Operators	YES	—
A.4	Keywords	YES	—
A.5	Terminals	YES	—
A.6	Formal EDDL syntax	YES	—

D.3 Profiles for FOUNDATION™⁷ Fieldbus

D.3.1 EDDL profile

Table D.6 is the selection of elements from the lexical structure constructs specified in Clause 7 and used by the consortium Fieldbus Foundation.

⁷ FOUNDATION™ Fieldbus is the trade name of the consortium Fieldbus Foundation (non-profit organization). This information is given for the convenience of users of this standard and does not constitute an endorsement by IEC of the trade name holder or any of its products. Compliance to this profile does not require use of the trade name. Use of the trade names requires permission from the trade name holder.

Table D.6 – EDDL element selection for Fieldbus Foundation

Reference	Basic construct	Attribute	Presence	Constraints
7.2.2.1		DD_REVISION	YES	—
7.2.2.2		DEVICE_REVISION	YES	—
7.2.2.3		DEVICE_TYPE	YES	—May not be an identifier; must be an integer.
7.2.2.4		EDD_PROFILE	NO	—
7.2.2.5		EDD_VERSION	NO	—
7.2.2.6		MANUFACTURER	YES	—May not be an identifier; must be an integer.
7.2.2.7		MANUFACTURER_EXT	NO	—
7.3	AXIS		YES	—
7.3.2.1		CONSTANT_UNIT	YES	—
7.36.8		HELP	YES	—
7.36.9		LABEL	YES	—
7.3.2.2		MAX_VALUE	YES	—
7.3.2.3		MIN_VALUE	YES	—
7.3.2.4		SCALING	YES	—
7.4.1	BLOCK_A		YES	—
7.4.1.2.1		CHARACTERISTICS	YES	—
7.36.9		LABEL	YES	—
7.4.1.2.2		PARAMETERS	YES	—
7.4.1.2.3		AXIS_ITEMS	YES	—
7.4.1.2.4		CHART_ITEMS	YES	—
7.4.1.2.5		COLLECTION_ITEMS	YES	—
7.4.1.2.6		EDIT_DISPLAY_ITEMS	YES	—
7.4.1.2.7		FILE_ITEMS	YES	—
7.4.1.2.8		GRAPH_ITEMS	YES	—
7.4.1.2.9		GRID_ITEMS	YES	—
7.4.1.2.10		IMAGE_ITEMS	YES	—
7.4.1.2.11		LIST_ITEMS	YES	—
7.4.1.2.12		LOCAL_PARAMETERS	YES	—
		HELP	YES	—
7.4.1.2.13		MENU_ITEMS	YES	—
7.4.1.2.14		METHOD_ITEMS	YES	—
7.4.1.2.15		PARAMETER_LISTS	YES	—
7.4.1.2.16		REFERENCE_ARRAY_ITEMS	YES	Referred as ITEM_ARRAY_ITEMS.
7.4.1.2.17		REFRESH_ITEMS	YES	—
7.4.1.2.18		SOURCE_ITEMS	YES	—
7.4.1.2.19		UNIT_ITEMS	YES	—
7.4.1.2.20		WAVEFORM_ITEMS	YES	—
7.4.1.2.21		WRITE_AS_ONE_ITEMS	YES	—
7.4.1.2.22		CHARTS	YES	—
7.4.1.2.23		LISTS	YES	—
7.4.1.2.24		GRAPHS	YES	—
7.4.1.2.25		GRIDS	YES	—
7.4.1.2.26		MENUS	YES	—
7.4.1.2.27		METHODS	YES	—
7.4.2	BLOCK_B		NO	—
7.5	CHART		YES	—
7.36.10		MEMBERS	YES	—
7.5.2.1		CYCLE_TIME	YES	—
7.36.7		HEIGHT	YES	—
7.36.8		HELP	YES	—
7.36.9		LABEL	YES	—
7.5.2.2		LENGTH	YES	—
7.5.2.3		TYPE	YES	—
7.36.16		VALIDITY	YES	—
7.36.17		WIDTH	YES	—
7.6	COLLECTION		YES	—
7.6.2		item-type	YES	If optional item-type specified, only: COLLECTION, DOMAIN, EDIT_DISPLAY, MENU, METHOD, PROGRAM, RECORD, REFERENCE_ARRAY, REFRESH, RESPONSE_CODES, UNIT, VALUE_ARRAY, VARIABLE, VARIABLE_LIST, WRITE_AS_ONE
7.36.10		MEMBERS	YES	—
7.36.8		HELP	YES	—

Reference	Basic construct	Attribute	Presence	Constraints
7.36.9		LABEL	YES	—
7.36.16		VALIDITY	NO	—
7.7	COMMAND		NO	—
7.8	COMPONENT		NO	—
7.9	COMPONENT_FOLD ER		NO	—
7.10	COMPONENT_REFE RENCE		NO	—
7.11	COMPONENT_RELA TION		NO	—
7.12	CONNECTION		NO	—
7.13	DOMAIN		YES	—
7.36.6		HANDLING	YES	—
7.36.14		RESPONSE_CODES	YES	Only: referenced RESPONSE_CODES
7.14	EDIT_DISPLAY		YES	—
7.14.2.1		EDIT_ITEMS	YES	—
7.36.9		LABEL	YES	—
7.14.2.2		DISPLAY_ITEMS	YES	—
7.14.2.3		POST_EDIT_ACTIONS	YES	Except: DEFINITION
7.14.2.4		PRE_EDIT_ACTIONS	YES	Except: DEFINITION
7.15	FILE		YES	—
7.36.12		MEMBERS	YES	—
7.36.8		HELP	YES	—
7.36.9		LABEL	YES	—
7.16	GRAPH		YES	—
7.36.12		MEMBERS	YES	—
7.16.2.1		CYCLE_TIME	YES	—
7.36.7		HEIGHT	YES	—
7.36.8		HELP	YES	—
7.36.9		LABEL	YES	—
7.36.16		VALIDITY	YES	—
7.36.17		WIDTH	YES	—
7.16.2.2		X_AXIS	YES	—
7.17	GRID		YES	—
7.17.2.1		VECTORS	YES	—
7.36.6		HANDLING	YES	—
7.36.7		HEIGHT	YES	—
7.36.8		HELP	YES	—
7.36.9		LABEL	YES	—
7.17.2.2		ORIENTATION	YES	—
7.36.16		VALIDITY	YES	—
7.36.17		WIDTH	YES	—
7.18	IMAGE		YES	—
7.18.2.1		PATH	YES	—
7.36.8		HELP	YES	—
7.36.9		LABEL	YES	—
7.18.2.2		LINK	YES	—
7.36.16		VALIDITY	YES	—
7.19	IMPORT		YES	—
7.20	INTERFACE		NO	—
7.21	LIKE		YES	The optional item-type shall be used
7.22	LIST		YES	—
7.22.2.1		TYPE	YES	—
7.22.2.2		CAPACITY	YES	—
7.22.2.3		COUNT	YES	—
7.36.8		HELP	YES	—
7.36.9		LABEL	YES	—
7.36.16		VALIDITY	YES	—
7.23	MENU		YES	—
7.23.2.1		ITEMS	YES	—
7.36.9		LABEL	YES	—
7.23.2.2		ACCESS	NO	—
7.36.8		HELP	NO	—
7.23.2.3		POST_EDIT_ACTIONS	NO	Except: DEFINITION
7.23.2.4		POST_READ_ACTIONS	NO	Except: DEFINITION
7.23.2.5		POST_WRITE_ACTIONS	NO	Except: DEFINITION
7.23.2.6		PRE_EDIT_ACTIONS	NO	Except: DEFINITION
7.23.2.7		PRE_READ_ACTIONS	NO	Except: DEFINITION
7.23.2.8		PRE_WRITE_ACTIONS	NO	Except: DEFINITION
7.23.2.9		PURPOSE	NO	—

Reference	Basic construct	Attribute	Presence	Constraints
7.23.2.10		ROLE	NO	—
7.23.2.11		STYLE	YES	Except: string
7.36.16		VALIDITY	YES	—
7.24	METHOD		YES	—
7.36.4		DEFINITION	YES	—
7.24.2.1		ACCESS	NO	—
7.24.2.2		CLASS	YES	—
7.36.9		LABEL	YES	—
7.36.8		HELP	YES	—
7.24.2.3		TYPE	NO	—
7.36.16		VALIDITY	YES	—
7.25	PROGRAM		YES	—
7.25.2		ARGUMENTS	YES	—
7.36.13		RESPONSE_CODES	YES	Only: referenced RESPONSE_CODES
7.26	RECORD		YES	—
7.36.10		MEMBERS	YES	—
7.36.9		LABEL	YES	—
7.36.8		HELP	YES	—
7.36.14		RESPONSE_CODES	YES	Only: referenced RESPONSE_CODES
7.36.16		VALIDITY	YES	—
7.27	REFERENCE_ARRAY		YES	Referred as ITEM_ARRAY
7.6.2		item-type	YES	Only: COLLECTION, DOMAIN, EDIT_DISPLAY, MENU, METHOD, PROGRAM, RECORD, REFRESH, REFERENCE_ARRAY, REFRESH, RESPONSE_CODES, UNIT, VALUE_ARRAY, VARIABLE, VARIABLE_LIST, WRITE_AS_ONE
7.27.2		ELEMENTS	YES	—
7.36.8		HELP	YES	—
7.36.9		LABEL	YES	—
7.36.16		VALIDITY	YES	—
7.28.1	REFRESH		YES	—
7.28.2	UNIT		YES	—
7.28.3	WRITE_AS_ONE		YES	—
7.36.14	RESPONSE_CODES		YES	—
7.30	SOURCE		YES	—
7.36.12		MEMBERS	YES	—
7.36.5		EMPHASIS	YES	—
7.30.2.1		EXIT_ACTIONS	YES	—
7.36.8		HELP	YES	—
7.30.2.2		INIT_ACTIONS	YES	—
7.36.9		LABEL	YES	—
7.36.10		LINE_COLOR	YES	—
7.36.11		LINE_TYPE	YES	—
7.30.2.3		REFRESH_ACTIONS	YES	—
7.36.16		VALIDITY	YES	—
7.30.2.4		Y_AXIS	YES	—
7.31	TEMPLATE		NO	—
7.30.2.4	VALUE_ARRAY		YES	—
7.36.9		LABEL	YES	—
7.32.2.1		NUMBER_OF_ELEMENTS	YES	Must evaluate to a constant value.
7.32.2.2		TYPE	YES	—
7.36.8		HELP	YES	—
7.36.14		RESPONSE_CODES	YES	Only: referenced RESPONSE_CODES.
7.36.16		VALIDITY	YES	—
7.33	VARIABLE		YES	—
7.33.2.1		CLASS	YES	Every VARIABLE shall have one and only one of the following classes: CONTAINED, INPUT, OUTPUT Additional classes can be combined: ALARM, DIAGNOSTIC, DYNAMIC, LOCAL, OPERATE, SERVICE, TUNE
7.36.9		LABEL	YES	—
7.33.2.2		TYPE	YES	Except: DATE, OBJECT REFERENCE, PACKED_ASCII. No options on date and time types.
7.33.2.3		CONSTANT_UNIT	YES	—
7.33.2.4		DEFAULT_VALUE	NO	—
7.36.6		HANDLING	YES	—
7.36.8		HELP	YES	—

Reference	Basic construct	Attribute	Presence	Constraints
7.33.2.5		INITIAL_VALUE	NO	—
7.33.2.6		POST_EDIT_ACTIONS	YES	Except: DEFINITION
7.33.2.7		POST_READ_ACTIONS	YES	Except: DEFINITION
7.33.2.8		POST_WRITE_ACTIONS	YES	Except: DEFINITION
7.33.2.9		PRE_EDIT_ACTIONS	YES	Except: DEFINITION
7.33.2.10		PRE_READ_ACTIONS	YES	Except: DEFINITION
7.33.2.11		PRE_WRITE_ACTIONS	YES	Except: DEFINITION
7.33.2.12		READ_TIMEOUT	YES	—
7.33.2.12		WRITE_TIMEOUT	YES	—
7.33.2.13		REFRESH_ACTIONS	YES	—
7.36.13		RESPONSE_CODE	YES	Only: referenced RESPONSE_CODES.
7.33.2.13		STYLE	NO	—
7.36.16		VALIDITY	YES	—
7.34	VARIABLE_LIST		YES	—
7.36.10		MEMBERS	YES	—
7.36.8		HELP	YES	—
7.36.9		LABEL	YES	—
7.36.13		RESPONSE_CODES	YES	Only: referenced RESPONSE_CODES.
7.35	WAVEFORM		YES	—
7.35.2.1		TYPE	YES	—
7.36.5		EMPHASIS	YES	—
7.35.2.2		EXIT_ACTIONS	YES	—
7.36.6		HANDLING	YES	—
7.36.8		HELP	YES	—
7.35.2.3		INIT_ACTIONS	YES	—
7.35.2.4		KEY_POINTS	YES	—
7.36.9		LABEL	YES	—
7.36.10		LINE_COLOR	YES	—
7.36.11		LINE_TYPE	YES	—
7.35.2.5		REFRESH_ACTIONS	YES	—
7.36.16		VALIDITY	YES	—
7.35.2.6		Y_AXIS	YES	—
	Conditional expressions		YES	—
7.38	References		YES	Except: referencing HELP and LABEL attributes (see 7.39.5) of EDD instances.
7.38.27	Strings		YES	string operations (see 7.39.6) are not possible.
7.39.7	Expressions		YES	Except: ARRAY_INDEX (see Table 244) and attribute values of VARIABLES: SCALING_FACTOR, DEFAULT_VALUE, INITIAL_VALUE, VARIABLE_STATUS (see Table 245) are not possible.
7.41	Text dictionary		YES	—

D.3.2 Builtin profile

Table D.7 is the selection of Builtins specified in Annex B and used by the consortium Fieldbus Foundation.

The Builtins are listed in alphabetical order.

Table D.7 – Builtin profile for Fieldbus Foundation

Reference	Builtin Name	Presence	Constraints
B.3	ERROR	NO	—
B.4	TRACE	NO	—
B.5	WARNING	NO	—
B.6	abort	NO	—
B.7	abort_on_all_comm_errors	YES	—
B.8	ABORT_ON_ALL_COMM_STATUS	NO	—
B.9	ABORT_ON_ALL_DEVICE_STATUS	NO	—
B.10	ABORT_ON_ALL_RESPONSE_CODES	NO	—
B.11	abort_on_all_response_codes	YES	—
B.12	ABORT_ON_COMM_ERROR	NO	—
B.13	abort_on_comm_error	YES	—
B.14	ABORT_ON_COMM_STATUS	NO	—
B.15	ABORT_ON_DEVICE_STATUS	NO	—
B.16	ABORT_ON_NO_DEVICE	NO	—
B.17	ABORT_ON_RESPONSE_CODE	NO	—

Reference	Builtin Name	Presence	Constraints
B.18	abort_on_response_code	YES	—
B.19	abs	NO	—
B.20	ACKNOWLEDGE	NO	—
B.21	acknowledge	NO	—
B.22	acos	NO	—
B.23	add_abort_method (version A)	NO	—
B.24	add_abort_method (version B)	YES	—
B.25	AddTime	NO	—
B.26	asin	NO	—
B.27	assign	YES	—
B.28	assign_double	NO	—
B.29	assign_float	NO	—
B.30	assign_int	NO	—
B.31	assign_var	NO	—
B.32	assign2	YES	—
B.33	atan	NO	—
B.34	ATOF	NO	—
B.35	atof	NO	—
B.36	ATOI	NO	—
B.37	atoi	NO	—
B.38	BUILD_MESSAGE	NO	—
B.39	ByteToDouble	NO	—
B.40	ByteToFloat	NO	—
B.41	ByteToLong	NO	—
B.42	ByteToShort	NO	—
B.43	cbrt	NO	—
B.44	ceil	NO	—
B.45	cos	NO	—
B.46	cosh	NO	—
B.47	dassign	NO	—
B.48	DATE_AND_TIME_VALUE to string	NO	—
B.49	Date_to_DayOfMonth	NO	—
B.50	DATE_to_days	NO	—
B.51	Date_to_Month	NO	—
B.52	DATE_to_string	NO	—
B.53	Date_To_Time	NO	—
B.54	Date_to_Year	NO	—
B.55	days_to_DATE	NO	—
B.56	DELAY	NO	—
B.57	delay	NO	—
B.58	DELAY_TIME	NO	—
B.59	delayfor	YES	—
B.60	delayfor2	YES	—
B.61	DICT_ID	YES	—
B.62	dictionary_string	NO	—
B.63	DiffTime	NO	—
B.64	discard_on_exit	YES	—
B.65	DISPLAY	NO	—
B.66	display	NO	—
B.67	display_bitenum	NO	—
B.68	display_builtin_error	YES	—
B.69	display_comm_error	YES	—
B.70	display_comm_status	NO	—
B.71	display_device_status	NO	—
B.72	display_dynamics	YES	—
B.73	display_dynamics2	YES	—
B.74	display_message	YES	—
B.75	display_message2	YES	—
B.76	display_response_code	YES	—
B.77	display_response_status	NO	—
B.78	display_xmtr_status	NO	—
B.79	DoubleToByte	NO	—
B.80	edit_device_value	YES	—
B.81	edit_device_value2	YES	—
B.82	edit_local_value	YES	—
B.83	edit_local_value2	YES	—
B.84	exp	NO	—
B.85	ext_send_command	NO	—
B.86	ext_send_command_trans	NO	—
B.87	fail_on_all_comm_errors	YES	—
B.88	fail_on_all_response_codes	YES	—
B.89	fail_on_comm_error	YES	—

Reference	Builtin Name	Presence	Constraints
B.90	fail_on_response_code	YES	—
B.91	fassign	NO	—
B.92	fgetval	NO	—
B.93	float_value	NO	—
B.94	FloatToByte	NO	—
B.95	floor	NO	—
B.96	fmod	NO	—
B.97	fpclassify	NO	—
B.98	From_DATE_AND_TIME_VALUE	NO	—
B.99	From_TIME_VALUE	NO	—
B.100	fsetval	NO	—
B.101	ftoa	NO	—
B.102	fvar_value	NO	—
B.103	get_acknowledgement	YES	—
B.104	get_acknowledgement2	YES	—
B.105	get_block_instance_by_object_index	YES	—
B.106	get_block_instance_by_tag	YES	—
B.107	get_block_instance_count	YES	—
B.108	get_comm_error	YES	—
B.109	get_comm_error_string	YES	—
B.110	get_date	YES	—
B.111	get_date_lelem	YES	—
B.112	get_date_lelem2	YES	—
B.113	get_date_value	YES	—
B.114	get_date_value2	YES	—
B.115	GET_DD_REVISION	NO	—
B.116	get_dd_error	YES	—
B.117	GET_DEV_VAR_VALUE	NO	—
B.118	get_dev_var_value	NO	—
B.119	GET_DEVICE_REVISION	NO	—
B.120	GET_DEVICE_TYPE	NO	—
B.121	get_dictionary_string	NO	—
B.122	get_double	YES	—
B.123	get_double_lelem	YES	—
B.124	get_double_lelem2	YES	—
B.125	get_double_value	YES	—
B.126	get_double_value2	YES	—
B.127	get_enum_string	NO	—
B.128	get_float	YES	—
B.129	get_float_lelem	YES	—
B.130	get_float_lelem2	YES	—
B.131	get_float_value	YES	—
B.132	get_float_value2	YES	—
B.133	GET_LOCAL_VAR_VALUE	NO	—
B.134	get_local_var_value	NO	—
B.135	GET_MANUFACTURER	NO	—
B.136	get_more_status	NO	—
B.137	get_resolve_status	YES	—
B.138	get_response_code	YES	—
B.139	get_response_code_string	YES	—
B.140	get_rspcode_string	NO	—
B.141	get_rspcode_string_by_id	NO	—
B.142	get_signed	YES	—
B.143	get_signed_lelem	YES	—
B.144	get_signed_lelem2	YES	—
B.145	get_signed_value	YES	—
B.146	get_signed_value2	YES	—
B.147	get_status_code_string	NO	—
B.148	get_status_string	YES	—
B.149	get_stddict_string	YES	—
B.150	get_string	YES	—
B.151	get_string_lelem	YES	—
B.152	get_string_lelem2	YES	—
B.153	get_string_value	YES	—
B.154	get_string_value2	YES	—
B.155	GET_TICK_COUNT	NO	—
B.156	get_unsigned	YES	—
B.157	get_unsigned_lelem	YES	—
B.158	get_unsigned_lelem2	YES	—
B.159	get_unsigned_value	YES	—
B.160	get_unsigned_value2	YES	—
B.161	get_variable_string	NO	—

Reference	Builtin Name	Presence	Constraints
B.162	GetCurrentDate	NO	—
B.163	GetCurrentDateAndTime	NO	—
B.164	GetCurrentTime	NO	—
B.165	iassign	NO	—
B.166	igetval	NO	—
B.167	IGNORE_ALL_COMM_STATUS	NO	—
B.168	IGNORE_ALL_DEVICE_STATUS	NO	—
B.169	IGNORE_ALL_RESPONSE_CODES	NO	—
B.170	IGNORE_COMM_ERROR	NO	—
B.171	IGNORE_COMM_STATUS	NO	—
B.172	IGNORE_DEVICE_STATUS	NO	—
B.173	IGNORE_NO_DEVICE	NO	—
B.174	IGNORE_RESPONSE_CODE	NO	—
B.175	int_value	NO	—
B.176	is_NaN	YES	—
B.177	isetval	NO	—
B.178	ITEM_ID	YES	—
B.179	ITOA	NO	—
B.180	itoa (version A)	NO	—
B.181	itoa (version B)	NO	—
B.182	ivar_value	NO	—
B.183	lassign	NO	—
B.184	lgetval	NO	—
B.185	ListDeleteElementAt	YES	—
B.186	ListDeleteElementAt2	YES	—
B.187	ListInsert	YES	—
B.188	ListInsert2	YES	—
B.189	log	NO	—
B.190	LOG_MESSAGE	NO	—
B.191	log10	NO	—
B.192	log2	NO	—
B.193	long_value	NO	—
B.194	LongToByte	NO	—
B.195	lsetval	NO	—
B.196	lvar_value	NO	—
B.197	Make_Time	NO	—
B.198	MEMBER_ID	YES	—
B.199	MenuDisplay	YES	—
B.200	method_abort	YES	—
B.201	nan	NO	—
B.202	NaN_value	YES	—
B.203	nanf	NO	—
B.204	ObjectReference	NO	—
B.205	pop_abort_method	NO	—
B.206	pow	NO	—
B.207	process_abort	NO	—
B.208	push_abort_method	NO	—
B.209	put_date	YES	—
B.210	put_date_value	YES	—
B.211	put_date_value2	YES	—
B.212	put_double	YES	—
B.213	put_double_value	YES	—
B.214	put_double_value2	YES	—
B.215	put_float	YES	—
B.216	put_float_value	YES	—
B.217	put_float_value2	YES	—
B.218	PUT_MESSAGE	NO	—
B.219	put_message	NO	—
B.220	put_signed	YES	—
B.221	put_signed_value	YES	—
B.222	put_signed_value2	YES	—
B.223	put_string	YES	—
B.224	put_string_value	YES	—
B.225	put_string_value2	YES	—
B.226	put_unsigned	YES	—
B.227	put_unsigned_value	YES	—
B.228	put_unsigned_value2	YES	—
B.229	read_value	YES	—
B.230	read_value2	YES	—
B.231	ReadCommand	NO	—
B.232	remove_abort_method (version A)	NO	—
B.233	remove_abort_method (version B)	YES	—

Reference	Builtin Name	Presence	Constraints
B.234	remove_all_abort_methods	YES	—
B.235	resolve_array_ref	YES	—
B.236	resolve_block_ref	YES	—
B.237	resolve_list_ref	YES	—
B.238	resolve_local_ref	YES	—
B.239	resolve_local_ref2	YES	—
B.240	resolve_param_list_ref	YES	—
B.241	resolve_param_ref	YES	—
B.242	resolve_param_ref2	YES	—
B.243	resolve_record_ref	YES	—
B.244	resolve_record_ref2	YES	—
B.245	ret_double_value	YES	—
B.246	ret_double_value2	YES	—
B.247	ret_float_value	YES	—
B.248	ret_float_value2	YES	—
B.249	ret_signed_value	YES	—
B.250	ret_signed_value2	YES	—
B.251	ret_unsigned_value	YES	—
B.252	ret_unsigned_value2	YES	—
B.253	retry_on_all_comm_errors	YES	—
B.254	RETRY_ON_ALL_COMM_STATUS	NO	—
B.255	RETRY_ON_ALL_DEVICE_STATUS	NO	—
B.256	RETRY_ON_ALL_RESPONSE_CODES	NO	—
B.257	retry_on_all_response_codes	YES	—
B.258	RETRY_ON_COMM_ERROR	NO	—
B.259	retry_on_comm_error	YES	—
B.260	RETRY_ON_COMM_STATUS	NO	—
B.261	RETRY_ON_DEVICE_STATUS	NO	—
B.262	RETRY_ON_NO_DEVICE	NO	—
B.263	RETRY_ON_RESPONSE_CODE	NO	—
B.264	retry_on_response_code	YES	—
B.265	round	NO	—
B.266	save_on_exit	YES	—
B.267	save_values	NO	—
B.268	seconds_to_TIME_VALUE	NO	—
B.269	SELECT_FROM_LIST	NO	—
B.270	select_from_list	NO	—
B.271	select_from_menu	YES	—
B.272	select_from_menu2	YES	—
B.273	send	NO	—
B.274	send_all_values	YES	—
B.275	send_command	NO	—
B.276	send_command_trans	NO	—
B.277	send_on_exit	YES	—
B.278	send_trans	NO	—
B.279	send_value	YES	—
B.280	send_value2	YES	—
B.281	SET_NUMBER_OF_RETRIES	NO	—
B.282	sgenval	NO	—
B.283	ShortToByte	NO	—
B.284	sin	NO	—
B.285	sinh	NO	—
B.286	sqrt	NO	—
B.287	ssetval	NO	—
B.288	strcmp	NO	—
B.289	strleft	NO	—
B.290	strlen	NO	—
B.291	strlwr	NO	—
B.292	strmid	NO	—
B.293	strright	NO	—
B.294	strstr	NO	—
B.295	strtrim	NO	—
B.296	strupr	NO	—
B.297	tan	NO	—
B.298	tanh	NO	—
B.299	Time_To_Date	NO	—
B.300	TIME_VALUE_to_Hour	NO	—
B.301	TIME_VALUE_to_Minute	NO	—
B.302	TIME_VALUE_to_Second	NO	—
B.303	TIME_VALUE_to_seconds	NO	—
B.304	TIME_VALUE_to_string	NO	—
B.305	timet_to_string	NO	—

Reference	Builtin Name	Presence	Constraints
B.306	timet_to_TIME_VALUE	NO	—
B.307	To Date	NO	—
B.308	To Date_and_Time	NO	—
B.309	To Time	NO	—
B.310	To_TIME_VALUE	NO	—
B.311	trunc	NO	—
B.312	VARID	NO	—
B.313	vassign	NO	—
B.314	WriteCommand	NO	—
B.315	XMTR_ABORT_ON_ALL_COMM_STATUS	NO	—
B.316	XMTR_ABORT_ON_ALL_DATA	NO	—
B.317	XMTR_ABORT_ON_ALL_DEVICE_STATUS	NO	—
B.318	XMTR_ABORT_ON_ALL_RESPONSE_CODES	NO	—
B.319	XMTR_ABORT_ON_COMM_ERROR	NO	—
B.320	XMTR_ABORT_ON_COMM_STATUS	NO	—
B.321	XMTR_ABORT_ON_DATA	NO	—
B.322	XMTR_ABORT_ON_DEVICE_STATUS	NO	—
B.323	XMTR_ABORT_ON_NO_DEVICE	NO	—
B.324	XMTR_ABORT_ON_RESPONSE_CODE	NO	—
B.325	XMTR_IGNORE_ALL_COMM_STATUS	NO	—
B.326	XMTR_IGNORE_ALL_DATA	NO	—
B.327	XMTR_IGNORE_ALL_DEVICE_STATUS	NO	—
B.328	XMTR_IGNORE_ALL_RESPONSE_CODES	NO	—
B.329	XMTR_IGNORE_COMM_ERROR	NO	—
B.330	XMTR_IGNORE_COMM_STATUS	NO	—
B.331	XMTR_IGNORE_DEVICE_STATUS	NO	—
B.332	XMTR_IGNORE_NO_DEVICE	NO	—
B.333	XMTR_IGNORE_RESPONSE_CODE	NO	—
B.334	XMTR_RETRY_ON_ALL_COMM_STATUS	NO	—
B.335	XMTR_RETRY_ON_ALL_DATA	NO	—
B.336	XMTR_RETRY_ON_ALL_DEVICE_STATUS	NO	—
B.337	XMTR_RETRY_ON_ALL_RESPONSE_CODE	NO	—
B.338	XMTR_RETRY_ON_ALL_RESPONSE_CODES	NO	—
B.339	XMTR_RETRY_ON_COMM_ERROR	NO	—
B.340	XMTR_RETRY_ON_COMM_STATUS	NO	—
B.341	XMTR_RETRY_ON_DATA	NO	—
B.342	XMTR_RETRY_ON_DEVICE_STATUS	NO	—
B.343	XMTR_RETRY_ON_NO_DEVICE	NO	—
B.344	XMTR_RETRY_ON_RESPONSE_CODE	NO	—
B.345	Return Codes	YES	—

D.3.3 EDDL Formal Definition profile

Reference	Title	Presence	Constraints
A.1	EDDL preprocessor	YES	—
A.2	Conventions	YES	—
A.3	Operators	YES	—
A.4	Keywords	YES	—
A.5	Terminals	YES	—
A.6	Formal EDDL syntax	YES	—

D.4 Profiles for HART® Communication Foundation (HCF)⁸

D.4.1 EDDL profile

Table D.8 is the selection of elements from the lexical structure constructs specified in Clause 7 and used by the consortium HART ® Communication Foundation.

Table D.8 – EDDL element selection for HCF

Reference	Basic construct	Attribute	Presence	Constraints
7.2.2.1	DD_REVISION		YES	One octet.
7.2.2.2	DEVICE_REVISION		YES	One octet.

⁸ HART® is a registered trademark of the consortium HART Communication Foundation (HCF) (non-profit organization). This information is given for the convenience of users of this standard and does not constitute an endorsement by IEC of the trade name holder or any of its products. Compliance to this profile does not require use of the trade name. Use of the trade names requires permission from the trade name holder.

Reference	Basic construct	Attribute	Presence	Constraints
7.2.2.3	DEVICE_TYPE		YES	Two octets.
7.2.2.4	EDD_PROFILE		NO	—
7.2.2.5	EDD_VERSION		NO	—
7.2.2.6	MANUFACTURER		YES	Two octets.
7.2.2.7	MANUFACTURER_EX T		NO	—
7.3	AXIS		YES	Only: CONSTANT_UNIT, HELP, LABEL, MAX_VALUE, MIN_VALUE, SCALING
7.3.2.1		CONSTANT_UNIT	YES	Not Conditional
7.4	BLOCK		NO	—
7.5	CHART		YES	Only: LABEL, HELP, VALIDITY, HEIGHT, WIDTH, TYPE, LENGTH, CYCLE_TIME, MEMBERS
7.5.2.1		CYCLE_TIME	YES	only constant-expression
7.5.2.2		LENGTH	YES	only constant-expression
7.6	COLLECTION		YES	Only: LABEL, HELP, VALIDITY, MEMBERS
7.6.2		item-type	YES	Use is discouraged. Supported for backward compatibility. Only item-types: VARIABLE, ARRAY, COLLECTION, LIST, MENU, EDIT_DISPLAY, GRAPH, CHART, COMMAND, METHOD, FILE
7.7	COMMAND		YES	Only: NUMBER, OPERATION, TRANSACTION, RESPONSE_CODE
7.7.2.1		OPERATION	YES	Only: READ, WRITE, COMMAND, no conditionals.
7.7.2.2		TRANSACTION	YES	No conditionals. Request and reply may only contain variable references; lists, array, collections not supported.
7.7.2.5		NUMBER	YES	only constant-expression.
7.36.14		RESPONSE_CODES	YES	Only: embedded RESPONSE_CODES; no conditionals.
7.8	COMPONENT		NO	—
7.9	COMPONENT_FOLDE R		NO	—
7.10	COMPONENT_REFER ENCE		NO	—
7.11	COMPONENT_RELATI ON		NO	—
7.12	CONNECTION		NO	—
7.13	DOMAIN		NO	—
7.14	EDIT_DISPLAY		YES	Only: LABEL, HELP, VALIDITY, DISPLAY_ITEMS, EDIT_ITEMS, PRE_EDIT_ACTIONS, POST_EDIT_ACTIONS
7.14.2.1		EDIT_ITEMS	YES	Only: VARIABLE, WRITE_AS_ONE
7.14.2.2		DISPLAY_ITEMS	YES	Only: VARIABLE
7.14.2.3		POST_EDIT_ACTIONS	YES	Except: DEFINITION
7.14.2.4		PRE_EDIT_ACTIONS	YES	Except: DEFINITION
7.15	FILE		YES	Only: LABEL, HELP, MEMBERS
7.16	GRAPH		YES	Only: LABEL, HELP, VALIDITY, HEIGHT, WIDTH, CYCLE_TIME, X_AXIS, MEMBERS
7.17	GRID		YES	Only: LABEL, HELP, VALIDITY, HEIGHT, WIDTH, ORIENTATION, HANDLING, VECTORS
7.18	IMAGE		YES	Only: LABEL, HELP, VALIDITY, LINK, PATH
7.19	IMPORT		YES	EDD cannot be referenced by file name
7.20	INTERFACE		NO	—
7.21	LIKE		YES	—
7.22	LIST		YES	Only: LABEL, HELP, VALIDITY, CAPACITY, COUNT, TYPE
7.23	MENU		YES	Only: LABEL, HELP, VALIDITY, STYLE, ITEMS
7.23.2.1		ITEMS	YES	Only: menu, variable, write-as-one, edit-display, method, collection, graph, chart, image, grid, references plus constant-string, ROWBREAK and

Reference	Basic construct	Attribute	Presence	Constraints
				COLUMNBREAK For variable-reference qualifiers DISPLAY_VALUE READ_ONLY NO_LABEL NO_UNIT are supported For menu-reference REVIEW is supported. For image-reference INLINE and NO_LABEL are supported
7.24	METHOD		YES	Only: LABEL, HELP, VALIDITY, CLASS, TYPE, DEFINITION.
7.24.2.2		CLASS	YES	Only: ALARM, COMPUTATION, ANALOG_CHANNEL, CORRECTION, DEVICE, DISCRETE, FREQUENCY, HART, LOCAL_DISPLAY, MODE, RANGE, TUNE. CLASS is Optional attribute
7.25	PROGRAM		NO	—
7.26	RECORD		NO	—
7.27	REFERENCE_ARRAY		YES	Only: item-type, LABEL, HELP, VALIDITY, ELEMENTS Concrete Syntax: ARRAY.
7.28.1	REFRESH		YES	—
7.28.2	UNIT		YES	—
7.28.3	WRITE_AS_ONE		YES	—
7.36.14	RESPONSE_CODES		NO	—
7.30	SOURCE		YES	Only: LABEL, HELP, VALIDITY, EMPHASIS, LINE_TYPE, LINE_COLOR, Y_AXIS, MEMBERS, INIT_ACTIONS, REFRESH_ACTIONS, EXIT_ACTIONS,
7.31	TEMPLATE		NO	—
7.32	VALUE_ARRAY		YES	Only: LABEL, HELP, VALIDITY, TYPE, NUMBER_OF_ELEMENTS Concrete syntax: ARRAY
7.32.2.1		NUMBER_OF_ELEMENTS	YES	Must evaluate to a constant value.
7.33	VARIABLE		YES	Only: LABEL, HELP, CLASS, VALIDITY, HANDLING, CONSTANT_UNIT, DEFAULT_VALUE, TYPE, PRE_EDIT_ACTIONS, POST_EDIT_ACTIONS, PRE_READ_ACTIONS, POST_READ_ACTIONS, PRE_WRITE_ACTIONS, POST_WRITE_ACTIONS, REFRESH_ACTIONS, READ_TIMEOUT, WRITE_TIMEOUT
7.33.2.1		CLASS	YES	Optional attribute Only: ALARM, COMPUTATION, ANALOG_CHANNEL, CORRECTION, DEVICE, DISCRETE, FREQUENCY, HART, LOCAL_DISPLAY, MODE, RANGE, TUNE, If not CLASS LOCAL then can be combined with DYNAMIC, DIAGNOSTIC, SERVICE
7.36.9		LABEL	YES	Is optional
7.33.2.2		TYPE	YES	Only: FLOAT, DOUBLE, INTEGER, UNSIGNED_INTEGER, ENUMERATED, BIT_ENUMERATED, INDEX, PACKED_ASCII, ASCII, PASSWORD, DATE, TIME_VALUE, No conditionals type. No INITIAL_VALUE. Default value of INTEGER and UNSIGNED_INTEGER is 1. Initial value of all PACKED_ASCII, ASCII and PASSWORD is a string

Reference	Basic construct	Attribute	Presence	Constraints
				with each character set to '?'. No Conditionals
7.33.2.3		CONSTANT_UNIT	YES	Except: DEFINITION
7.33.2.6		POST_EDIT_ACTIONS	YES	Except: DEFINITION
7.33.2.7		POST_READ_ACTIONS	YES	Except: DEFINITION
7.33.2.8		POST_WRITE_ACTIONS	YES	Except: DEFINITION
7.33.2.9		PRE_EDIT_ACTIONS	YES	Except: DEFINITION
7.33.2.10		PRE_READ_ACTIONS	YES	Except: DEFINITION
7.33.2.11		PRE_WRITE_ACTIONS	YES	Except: DEFINITION
7.34	VARIABLE_LIST		NO	—
7.35	WAVEFORM		YES	Only: LABEL, HELP, VALIDITY, HANDLING, EMPHASIS, LINE_TYPE, LINE_COLOR, Y_AXIS, KEY_POINTS, TYPE, INIT_ACTIONS, REFRESH_ACTIONS, EXIT_ACTIONS
7.37	Conditional expressions		YES	
7.38	Referencing		YES	Only attribute references: DEFAULT_VALUE, VIEW_MIN, VIEW_MAX, MAX_VALUE, MAX_VALUE1, MAX_VALUE2, MAX_VALUE3, MAX_VALUE4, MAX_VALUE5, MAX_VALUE6, MAX_VALUE7, MAX_VALUE8, MAX_VALUE9, MIN_VALUE, MIN_VALUE1, MIN_VALUE2, MIN_VALUE3, MIN_VALUE4, MIN_VALUE5, MIN_VALUE6, MIN_VALUE7, MIN_VALUE8, MIN_VALUE9, COUNT, CAPACITY, HELP, LABEL, DEFAULT_VALUE, Plus FIRST and LAST with LIST
7.39	Strings		YES	Only: string literals, string variable-references, dictionary references, enumeration variable value references
7.40	Expressions		YES	Only: constant, parenthesized expression, variable reference, attribute values of VARIABLES (DEFAULT_VALUE, VIEW_MIN, VIEW_MAX, MAX_VALUE, MAX_VALUE1, MAX_VALUE2, MAX_VALUE3, MAX_VALUE4, MAX_VALUE5, MAX_VALUE6, MAX_VALUE7, MAX_VALUE8, MAX_VALUE9, MIN_VALUE, MIN_VALUE1, MIN_VALUE2, MIN_VALUE3, MIN_VALUE4, MIN_VALUE5, MIN_VALUE6, MIN_VALUE7, MIN_VALUE8, MIN_VALUE9), attribute values of an AXIS (VIEW_MIN, VIEW_MAX), attribute values of a LIST (FIRST, LAST, COUNT, CAPACITY)
7.41	Text dictionary		YES	Only: HART Standard Dictionary plus one per manufacturer.
NOTE Text is only ISO Latin-1 or or UTF-8. Katakana, (kt) is entered as Romaji and is a series of English characters. 2-3 Characters per Katakana symbol.				

D.4.2 Builtin profile

Table D.9 is the selection of Builtins specified in Annex B and used by the consortium HART ® Communication Foundation (HCF).

The Builtins are listed in alphabetical order.

Table D.9 – Builtin profile for HCF

Reference	Builtin Name	Presence	Constraints
B.3	_ERROR	NO	—
B.4	_TRACE	NO	—
B.5	_WARNING	NO	—
B.6	abort	YES	—
B.7	abort_on_all_comm_errors	NO	—
B.8	ABORT_ON_ALL_COMM_STATUS	YES	—
B.9	ABORT_ON_ALL_DEVICE_STATUS	YES	—
B.10	ABORT_ON_ALL_RESPONSE_CODES	YES	—
B.11	abort_on_all_response_codes	NO	—
B.12	ABORT_ON_COMM_ERROR	YES	—
B.13	abort_on_comm_error	NO	—
B.14	ABORT_ON_COMM_STATUS	YES	—
B.15	ABORT_ON_DEVICE_STATUS	YES	—
B.16	ABORT_ON_NO_DEVICE	YES	—
B.17	ABORT_ON_RESPONSE_CODE	YES	—
B.18	abort_on_response_code	NO	—
B.19	abs	YES	—
B.20	ACKNOWLEDGE	YES	—
B.21	acknowledge	YES	—
B.22	acos	YES	—
B.23	add_abort_method (version A)	YES	—
B.24	add_abort_method (version B)	NO	—
B.25	AddTime	YES	—
B.26	asin	YES	—
B.27	assign	NO	—
B.28	assign_double	YES	—
B.29	assign_float	YES	—
B.30	assign_int	YES	—
B.31	assign_var	YES	—
B.32	assign2	NO	—
B.33	atan	YES	—
B.34	ATOF	NO	—
B.35	atof	YES	—
B.36	atoi	NO	—
B.37	atoi	YES	—
B.38	BUILD_MESSAGE	YES	—
B.39	ByteToDouble	NO	—
B.40	ByteToFloat	NO	—
B.41	ByteToLong	NO	—
B.42	ByteToShort	NO	—
B.43	cbrt	YES	—
B.44	ceil	YES	—
B.45	cos	YES	—
B.46	cosh	YES	—
B.47	dassign	YES	—
B.48	DATE AND TIME VALUE_to_string	YES	—
B.49	Date_to_DayOfMonth	YES	—
B.50	DATE_to_days	YES	—
B.51	Date_to_Month	YES	—
B.52	DATE_to_string	YES	—
B.53	Date_To_Time	YES	—
B.54	Date_to_Year	YES	—
B.55	days_to_DATE	YES	—
B.56	DELAY	YES	—
B.57	delay	YES	—
B.58	DELAY_TIME	YES	—
B.59	delayfor	NO	—
B.60	delayfor2	NO	—
B.61	DICT_ID	NO	—
B.62	dictionary_string	YES	—
B.63	DiffTime	YES	—
B.64	discard_on_exit	YES	—
B.65	DISPLAY	YES	—
B.66	display	YES	—
B.67	display_bitenum	NO	—
B.68	display_builtin_error	NO	—
B.69	display_comm_error	NO	—
B.70	display_comm_status	YES	—
B.71	display_device_status	YES	—
B.72	display_dynamics	NO	—

Reference	Builtin Name	Presence	Constraints
B.73	display_dynamics2	NO	—
B.74	display_message	NO	—
B.75	display_message2	NO	—
B.76	display_response_code	NO	—
B.77	display_response_status	YES	—
B.78	display_xmtr_status	YES	—
B.79	DoubleToByte	NO	—
B.80	edit_device_value	NO	—
B.81	edit_device_value2	NO	—
B.82	edit_local_value	NO	—
B.83	edit_local_value2	NO	—
B.84	exp	YES	—
B.85	ext_send_command	YES	—
B.86	ext_send_command_trans	YES	—
B.87	fail_on_all_comm_errors	NO	—
B.88	fail_on_all_response_codes	NO	—
B.89	fail_on_comm_error	NO	—
B.90	fail_on_response_code	NO	—
B.91	fassign	YES	—
B.92	fgetval	YES	—
B.93	float_value	YES	—
B.94	FloatToByte	NO	—
B.95	floor	YES	—
B.96	fmod	YES	—
B.97	fpclassify	YES	—
B.98	From_DATE_AND_TIME_VALUE	YES	—
B.99	From_TIME_VALUE	YES	—
B.100	fsetval	YES	—
B.101	ftoa	NO	—
B.102	fvar_value	YES	—
B.103	get_acknowledgement	NO	—
B.104	get_acknowledgement2	NO	—
B.105	get_block_instance_by_object_index	NO	—
B.106	get_block_instance_by_tag	NO	—
B.107	get_block_instance_count	NO	—
B.108	get_comm_error	NO	—
B.109	get_comm_error_string	NO	—
B.110	get_date	NO	—
B.111	get_date_lelem	NO	—
B.112	get_date_lelem2	NO	—
B.113	get_date_value	NO	—
B.114	get_date_value2	NO	—
B.115	GET_DD_REVISION	NO	—
B.116	get_dds_error	NO	—
B.117	GET_DEV_VAR_VALUE	YES	—
B.118	get_dev_var_value	YES	—
B.119	GET_DEVICE_REVISION	NO	—
B.120	GET_DEVICE_TYPE	NO	—
B.121	get_dictionary_string	YES	—
B.122	get_double	NO	—
B.123	get_double_lelem	NO	—
B.124	get_double_lelem2	NO	—
B.125	get_double_value	NO	—
B.126	get_double_value2	NO	—
B.127	get_enum_string	YES	—
B.128	get_float	NO	—
B.129	get_float_lelem	NO	—
B.130	get_float_lelem2	NO	—
B.131	get_float_value	NO	—
B.132	get_float_value2	NO	—
B.133	GET_LOCAL_VAR_VALUE	YES	—
B.134	get_local_var_value	YES	—
B.135	GET_MANUFACTURER	NO	—
B.136	get_more_status	YES	—
B.137	get_resolve_status	NO	—
B.138	get_response_code	NO	—
B.139	get_response_code_string	NO	—
B.140	get_rspcode_string	YES	—
B.141	get_rspcode_string_by_id	NO	—
B.142	get_signed	NO	—
B.143	get_signed_lelem	NO	—
B.144	get_signed_lelem2	NO	—

Reference	Builtin Name	Presence	Constraints
B.145	get_signed_value	NO	—
B.146	get_signed_value2	NO	—
B.147	get_status_code_string	YES	—
B.148	get_status_string	NO	—
B.149	get_stddict_string	NO	—
B.150	get_string	NO	—
B.151	get_string_lelem	NO	—
B.152	get_string_lelem2	NO	—
B.153	get_string_value	NO	—
B.154	get_string_value2	NO	—
B.155	GET_TICK_COUNT	NO	—
B.156	get_unsigned	NO	—
B.157	get_unsigned_lelem	NO	—
B.158	get_unsigned_lelem2	NO	—
B.159	get_unsigned_value	NO	—
B.160	get_unsigned_value2	NO	—
B.161	get_variable_string	NO	—
B.162	GetCurrentDate	NO	—
B.163	GetCurrentDateAndTime	NO	—
B.164	GetCurrentTime	YES	—
B.165	iassign	YES	—
B.166	igetval	YES	—
B.167	IGNORE_ALL_COMM_STATUS	YES	—
B.168	IGNORE_ALL_DEVICE_STATUS	YES	—
B.169	IGNORE_ALL_RESPONSE_CODES	YES	—
B.170	IGNORE_COMM_ERROR	YES	—
B.171	IGNORE_COMM_STATUS	YES	—
B.172	IGNORE_DEVICE_STATUS	YES	—
B.173	IGNORE_NO_DEVICE	YES	—
B.174	IGNORE_RESPONSE_CODE	YES	—
B.175	int_value	YES	—
B.176	is_NaN	NO	—
B.177	isetval	YES	—
B.178	ITEM_ID	NO	—
B.179	ITOA	NO	—
B.180	itoa (version A)	YES	—
B.181	itoa (version B)	NO	—
B.182	ivar_value	YES	—
B.183	lassign	YES	—
B.184	lgetval	NO	—
B.185	ListDeleteElementAt	YES	—
B.186	ListDeleteElementAt2	NO	—
B.187	ListInsert	YES	—
B.188	ListInsert2	NO	—
B.189	log	YES	—
B.190	LOG_MESSAGE	NO	—
B.191	log10	YES	—
B.192	log2	YES	—
B.193	long_value	NO	—
B.194	LongToByte	NO	—
B.195	lsetval	NO	—
B.196	lvar_value	NO	—
B.197	Make_Time	YES	—
B.198	MEMBER_ID	NO	—
B.199	MenuDisplay	YES	—
B.200	method_abort	NO	—
B.201	nan	YES	—
B.202	NaN_value	NO	—
B.203	nanf	YES	—
B.204	ObjectReference	NO	—
B.205	pop_abort_method	YES	—
B.206	pow	YES	—
B.207	process_abort	YES	—
B.208	push_abort_method	YES	—
B.209	put_date	NO	—
B.210	put_date_value	NO	—
B.211	put_date_value2	NO	—
B.212	put_double	NO	—
B.213	put_double_value	NO	—
B.214	put_double_value2	NO	—
B.215	put_float	NO	—
B.216	put_float_value	NO	—

Reference	Builtin Name	Presence	Constraints
B.217	put_float_value2	NO	—
B.218	PUT_MESSAGE	YES	—
B.219	put_message	YES	—
B.220	put_signed	NO	—
B.221	put_signed_value	NO	—
B.222	put_signed_value2	NO	—
B.223	put_string	NO	—
B.224	put_string_value	NO	—
B.225	put_string_value2	NO	—
B.226	put_unsigned	NO	—
B.227	put_unsigned_value	NO	—
B.228	put_unsigned_value2	NO	—
B.229	read_value	NO	—
B.230	read_value2	NO	—
B.231	ReadCommand	NO	—
B.232	remove_abort_method (version A)	YES	—
B.233	remove_abort_method (version B)	NO	—
B.234	remove_all_abort_methods	YES	—
B.235	resolve_array_ref	NO	—
B.236	resolve_block_ref	NO	—
B.237	resolve_list_ref	NO	—
B.238	resolve_local_ref	NO	—
B.239	resolve_local_ref2	NO	—
B.240	resolve_param_list_ref	NO	—
B.241	resolve_param_ref	NO	—
B.242	resolve_param_ref2	NO	—
B.243	resolve_record_ref	NO	—
B.244	resolve_record_ref2	NO	—
B.245	ret_double_value	NO	—
B.246	ret_double_value2	NO	—
B.247	ret_float_value	NO	—
B.248	ret_float_value2	NO	—
B.249	ret_signed_value	NO	—
B.250	ret_signed_value2	NO	—
B.251	ret_unsigned_value	NO	—
B.252	ret_unsigned_value2	NO	—
B.253	retry_on_all_comm_errors	NO	—
B.254	RETRY_ON_ALL_COMM_STATUS	YES	—
B.255	RETRY_ON_ALL_DEVICE_STATUS	YES	—
B.256	RETRY_ON_ALL_RESPONSE_CODES	YES	—
B.257	retry_on_all_response_codes	NO	—
B.258	RETRY_ON_COMM_ERROR	YES	—
B.259	retry_on_comm_error	NO	—
B.260	RETRY_ON_COMM_STATUS	YES	—
B.261	RETRY_ON_DEVICE_STATUS	YES	—
B.262	RETRY_ON_NO_DEVICE	YES	—
B.263	RETRY_ON_RESPONSE_CODE	YES	—
B.264	retry_on_response_code	NO	—
B.265	round	YES	—
B.266	save_on_exit	NO	—
B.267	save_values	YES	—
B.268	seconds_to_TIME_VALUE	YES	—
B.269	SELECT FROM LIST	YES	—
B.270	select_from_list	YES	—
B.271	select_from_menu	NO	—
B.272	select_from_menu2	NO	—
B.273	send	YES	—
B.274	send_all_values	NO	—
B.275	send_command	YES	—
B.276	send_command_trans	YES	—
B.277	send_on_exit	NO	—
B.278	send_trans	YES	—
B.279	send_value	NO	—
B.280	send_value2	NO	—
B.281	SET_NUMBER_OF_RETRIES	YES	—
B.282	sgetval	YES	—
B.283	ShortToByte	NO	—
B.284	sin	YES	—
B.285	sinh	YES	—
B.286	sqrt	YES	—
B.287	ssetval	YES	—
B.288	strcmp	YES	—

Reference	Builtin Name	Presence	Constraints
B.289	strleft	NO	—
B.290	strlen	YES	—
B.291	strlwr	YES	—
B.292	strmid	YES	—
B.293	strright	NO	—
B.294	strstr	YES	—
B.295	strtrim	YES	—
B.296	strupr	YES	—
B.297	tan	YES	—
B.298	tanh	YES	—
B.299	Time_To_Date	YES	—
B.300	TIME_VALUE_to_Hour	YES	—
B.301	TIME_VALUE_to_Minute	YES	—
B.302	TIME_VALUE_to_Second	YES	—
B.303	TIME_VALUE_to_seconds	YES	—
B.304	TIME_VALUE_to_string	YES	—
B.305	timet_to_string	YES	—
B.306	timet_to_TIME_VALUE	YES	—
B.307	To_Date	YES	—
B.308	To_Date_and_Time	YES	—
B.309	To_Time	YES	—
B.310	To_TIME_VALUE	YES	—
B.311	trunc	YES	—
B.312	VARID	YES	—
B.313	vassign	YES	—
B.314	WriteCommand	NO	—
B.315	XMTR_ABORT_ON_ALL_COMM_STATUS	YES	—
B.316	XMTR_ABORT_ON_ALL_DATA	YES	—
B.317	XMTR_ABORT_ON_ALL_DEVICE_STATUS	YES	—
B.318	XMTR_ABORT_ON_ALL_RESPONSE_CODES	YES	—
B.319	XMTR_ABORT_ON_COMM_ERROR	YES	—
B.320	XMTR_ABORT_ON_COMM_STATUS	YES	—
B.321	XMTR_ABORT_ON_DATA	YES	—
B.322	XMTR_ABORT_ON_DEVICE_STATUS	YES	—
B.323	XMTR_ABORT_ON_NO_DEVICE	YES	—
B.324	XMTR_ABORT_ON_RESPONSE_CODE	YES	—
B.325	XMTR_IGNORE_ALL_COMM_STATUS	YES	—
B.326	XMTR_IGNORE_ALL_DATA	YES	—
B.327	XMTR_IGNORE_ALL_DEVICE_STATUS	YES	—
B.328	XMTR_IGNORE_ALL_RESPONSE_CODES	YES	—
B.329	XMTR_IGNORE_COMM_ERROR	YES	—
B.330	XMTR_IGNORE_COMM_STATUS	YES	—
B.331	XMTR_IGNORE_DEVICE_STATUS	YES	—
B.332	XMTR_IGNORE_NO_DEVICE	YES	—
B.333	XMTR_IGNORE_RESPONSE_CODE	YES	—
B.334	XMTR_RETRY_ON_ALL_COMM_STATUS	YES	—
B.335	XMTR_RETRY_ON_ALL_DATA	YES	—
B.336	XMTR_RETRY_ON_ALL_DEVICE_STATUS	YES	—
B.337	XMTR_RETRY_ON_ALL_RESPONSE_CODE	YES	—
B.338	XMTR_RETRY_ON_ALL_RESPONSE_CODES	YES	—
B.339	XMTR_RETRY_ON_COMM_ERROR	YES	—
B.340	XMTR_RETRY_ON_COMM_STATUS	YES	—
B.341	XMTR_RETRY_ON_DATA	YES	—
B.342	XMTR_RETRY_ON_DEVICE_STATUS	YES	—
B.343	XMTR_RETRY_ON_NO_DEVICE	YES	—
B.344	XMTR_RETRY_ON_RESPONSE_CODE	YES	—
B.345	Return Codes	YES	—

D.4.3 EDDL Formal Definition profile

Reference	Title	Presence	Constraints
A.1	EDDL preprocessor	YES	—
A.2	Conventions	YES	—
A.3	Operators	YES	—
A.4	Keywords	YES	—
A.5	Terminals	YES	—
A.6	Formal EDDL syntax	YES	—

D.5 Data types

D.5.1 METHOD DEFINITIONS data types

The specified size in brackets of the VARIABLE TYPES ensures that an overflow or underflow cannot occur. Table D.10 specifies the METHOD DEFINITIONS data types and Table D.11 specifies the VARIABLE TYPES data types.

Table D.10 – METHOD DEFINITIONS data types

data types	Minimum value	Maximum value	Remark
char	-1 28	+1 27	—
char[size]	-128 for each character	+127 for each character	—
char[]	-128 for each character	+127 for each character	Unspecified length of a character string.
short	-32 768	+32 767	—
int/ long	-2 147 483 648	+2 147 483 647	—
unsigned char	0	+255	—
unsigned short	0	+65 535	—
unsigned int / unsigned long	0	+4 294 967 295	—
float	-3,402 823 466 e + 38	+3,402 823 466 e + 38	—
double	-1,797 693 134 862 315 7 e + 308	+1,797 693 134 862 315 7 e + 308	—
time_t	not specified	not specified	Opaque data type whose internal structure is specified by EDD application.
DD_STRING	—	—	Arbitrary length character string.
DD_ITEM	0	+4 294 967 295	ID of any item in the EDD.

Table D.11 – VARIABLE TYPES

VARIABLE TYPES	Minimum value	Maximum value
DOUBLE	-1,797 693 134 862 315 7e + 308	+1,797 693 134 862 315 7e + 308
FLOAT	-3,402 823 466 e + 38	+3,402 823 466 e + 38
INTEGER(1)	-128	+127
INTEGER(2)	-32 768	+32 767
INTEGER(3)	-8 388 608	+8 388 607
IINTEGER(4)	-2 147 483 648	+2 147 483 647
UNSIGNED_INTEGER(1)	0	+255
UNSIGNED_INTEGER(2)	0	+65 535
UNSIGNED_INTEGER(3)	0	+16 777 215
UNSIGNED_INTEGER(4)	0	+4 294 967 295
DATE	see D.5.2	see D.5.2

VARIABLE TYPES	Minimum value	Maximum value
DATE_AND_TIME	see D.5.3	see D.5.3
DURATION	see D.5.4	see D.5.4
TIME	see D.5.5	see D.5.5
TIME_VALUE	see D.5.6	see D.5.6
BIT_ENUMERATED(1)	0	+255
BIT_ENUMERATED(2)	0	+65 535
BIT_ENUMERATED(3)	0	+16 777 215
BIT_ENUMERATED(4)	0	+4 294 967 295
ENUMERATED(1)	0	+255
ENUMERATED(2)	0	+65 535
ENUMERATED(3)	0	+16 777 215
ENUMERATED(4)	0	+4 294 967 295
INDEX(1)	0	+255
INDEX(2)	0	+65 535
INDEX(3)	0	+16 777 215
INDEX(4)	0	+4 294 967 295
OBJECT_REFERENCE	see 7.33.2.2.2.6	see 7.33.2.2.2.6
ASCII(size)	0	+255 per character
BITSTRING(size)	0	+255 per character
EUC(size)	0	+255 per character
OCTET(size)	0	+255 per character
PACKED_ASCII	see D.5.7	see D.5.7
PASSWORD(size)	0	+255 per character
VISIBLE(size)	0	+255 per character

D.5.2 Coding of data DATE

Table D.12 specifies the DATE coding.

Table D.12 – DATE coding

Octet	Bit								Description
	8	7	6	5	4	3	2	1	
1	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	1..31 day of month
2	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	1...12 month
3	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	1...255 years (actual year minus 1900)

D.5.3 Coding of data DATE_AND_TIME

Table D.13 specifies the DATE_AND_TIME coding.

Table D.13 – DATE_AND_TIME coding

Octet	Bit								Description
	8	7	6	5	4	3	2	1	
1	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	0...59 999 ms
2	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
3	R		2^5	2^4	2^3	2^2	2^1	2^0	0...59 min
4	S	R		2^4	2^3	2^2	2^1	2^0	0...23 h
5	2^2	2^1	2^0	2^4	2^3	2^2	2^1	2^0	Bits 6–8: 1..7 day of week (1=Monday, 7=Sunday) Bits 1–5: 1..31 day of month
6	R		2^5	2^4	2^3	2^2	2^1	2^0	1...12 months
7	R	2^6	2^5	2^4	2^3	2^2	2^1	2^0	1...255 years (actual year minus 1900)
R means reserved for future use; S = 0 indicates standard time; S = 1 indicates summer time									

D.5.4 Coding of data DURATION

Table D.14 specifies the DURATION coding.

Table D.14 – DURATION coding

Octet	Bit								Description
	8	7	6	5	4	3	2	1	
1	0	0	0	0	2^{27}	2^{26}	2^{25}	2^{24}	Number of ms since midnight.
2	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	
3	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	
4	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
5	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	Number of days.
6	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	

D.5.5 Coding of data TIME

Table D.15 specifies the TIME coding.

Table D.15 – TIME coding

Octet	Bit								Description
	8	7	6	5	4	3	2	1	
1	0	0	0	0	2^{27}	2^{26}	2^{25}	2^{24}	Number of milliseconds since midnight.
2	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	
3	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	
4	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
5	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	Number of days since 01.01.1984.
6	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	

D.5.6 Coding of data TIME_VALUE

Table D.16 specifies the four octet TIME_VALUE coding.

Table D.16 – TIME_VALUE coding (four octets)

Octet	Bit								Description
	8	7	6	5	4	3	2	1	
1	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	Number of 1/32 ms.
2	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	
3	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	
4	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	

Table D.17 specifies the eight octet TIME_VALUE coding.

Table D.17 – TIME_VALUE coding (eight octets)

Octet	Bit								Description
	8	7	6	5	4	3	2	1	
1	SN	2 ⁶²	2 ⁶¹	2 ⁶⁰	2 ⁵⁹	2 ⁵⁸	2 ⁵⁷	2 ⁵⁶	Number of 1/32 ms since January 1, 1972.
2	2 ⁵⁵	2 ⁵⁴	2 ⁵³	2 ⁵²	2 ⁵¹	2 ⁵⁰	2 ⁴⁹	2 ⁴⁸	
3	2 ⁴⁷	2 ⁴⁶	2 ⁴⁵	2 ⁴⁴	2 ⁴³	2 ⁴²	2 ⁴¹	2 ⁴⁰	
4	2 ³⁹	2 ³⁸	2 ³⁷	2 ³⁶	2 ³⁵	2 ³⁴	2 ³³	2 ³²	
5	2 ³¹	2 ³⁰	2 ²⁹	2 ²⁸	2 ²⁷	2 ²⁶	2 ²⁵	2 ²⁴	
6	2 ²³	2 ²²	2 ²¹	2 ²⁰	2 ¹⁹	2 ¹⁸	2 ¹⁷	2 ¹⁶	
7	2 ¹⁵	2 ¹⁴	2 ¹³	2 ¹²	2 ¹¹	2 ¹⁰	2 ⁹	2 ⁸	
8	2 ⁷	2 ⁶	2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰	
SN: 0 = data is positive; 1 = data is negative and in two's complement representation.									

D.5.7 Coding of PACKED_ASCII (6-BIT ASCII) DATA FORMAT

Some of the alpha-numeric data passed by the protocol is transmitted to and from the devices in the PACKED_ASCII format. PACKED_ASCII is a subset of ASCII produced by removing the two most significant bits of each ASCII character. This allows four PACKED_ASCII characters to be placed in the space of three ASCII characters. Typically, PACKED_ASCII strings are defined in even multiples of three octets, four ASCII characters.

Construction of PACKED_ASCII characters:

- Truncate Bit #6 and #7 of each ASCII character
- Pack four, 6 bit-ASCII characters into three octets

Reconstruction of ASCII characters:

- Unpack the four, 6-bit ASCII characters
- Place the complement of Bit #5 of each unpacked, 6-bit ASCII character into Bit #6
- Set Bit #7 of each of the unpacked ASCII characters to zero.

Table D.18 shows the coding of the PACKED_ASCII characters.

Table D.18 – PACKED_ASCII coding

Character	Code	Character	Code	Character	Code	Character	Code
@	00	P	10	Space	20	0	30
A	01	Q	11	!	21	1	31
B	02	R	12	"	22	2	32
C	03	S	13	#	23	3	33
D	04	T	14	\$	24	4	34

Character	Code	Character	Code	Character	Code	Character	Code
E	05	U	15	%	25	5	35
F	06	V	16	&	26	6	36
G	07	W	17	'	27	7	37
H	08	X	18	(	28	8	38
I	09	Y	19	)	29	9	39
J	0A	Z	1A	*	2A	:	3A
K	0B	[	1B	+	2B	;	3B
L	0C	\	1C	,	2C	<	3C
M	0D	]	1D	-	2D	=	3D
N	0E	^	1E	.	2E	>	3E
O	0F	_	1F	/	2F	?	3F

IECNORM.COM: Click to view the full PDF of IEC 61804-3:2010

Withdrawing

Annex E (informative)

Historical background

This standard contains the EDDL specified in IEC 61804-2:2004 and the enhancements defined in the DD-Cooperation⁹ document FF-644: EDDL Specification Revision 1.2 with the phase 1 enhancements. Subclauses 4.1 and 4.2, and Clauses 5, 6, 7, and 8 of IEC 61804-2:2004, dealing with FB, are unchanged in the second edition of IEC 61804-2; IEC 61804-2:2004, Clause 3 is now split into two clauses covering terms and definitions (new version of IEC 61804-3:2006, Clause 3) and conventions (new IEC 61804-3:2006, Clause 4).

Compared to IEC 61804-2:2004, this standard includes the following enhancements.

- Charting – Enables graphical display of real-time (actual) data from devices
 - New CHART construct to define display characteristics
 - New SOURCE construct enables multiples curves on a CHART
 - New AXIS construct
- Graphing – Enables graphical display of static Y-t and XY data
 - New GRAPH construct to define display characteristics
 - New WAVEFORM construct enables multiple curves on a GRAPH.
 - New AXIS construct
- Improved Data Storage – Enables DD Developer to securely store data on the host
 - New FILE construct describes parameters that will be stored
 - New LIST construct is used with FILE to access specific parameters
- Improved User Interface (UI) – DD Developer can describe screen layout
 - Enhanced MENU construct with screen layout attributes (for example, dialog boxes)
- New METHOD builtins – support graphing, charting, storage and UI enhancements
 - New PlotCreate creates a graph or chart that can then be manipulated
 - New PlotDisplay displays a graph or chart
 - New PlotEdit allows the user to edit a graph or chart
 - New PlotRefresh causes the plot to be re-displayed
 - New PlotDestroy deletes a graph or chart
 - New ListInsert inserts an element into a list
 - New ListDeleteElementAt deletes an element from a list

EDDL is the most important and widely used digital communication descriptive language in the manufacturing and process industries.

⁹ DD-cooperation is a cooperation among the consortia of Fieldbus Foundation, Hart Communication Foundation, PROFIBUS Nutzerorganisation e.V. and, since 2006, the OPC Foundation. This second edition of EDDL extends the capabilities of the Device Description Language (EDDL) of IEC 61804-2:2004 to provide an industry standard solution for advanced visualization of intelligent device information to maintain the proven integrity of existing EDD technology across all three communication technologies according to IEC 61784-1, CPF 1, IEC 61784-1, CP 3/1 and IEC 61784-1, CP 3/2, and IEC 61784-1, CPF 9.

Bibliography

IEC 60050-351:1998, *International Electrotechnical Vocabulary (IEV) – Part 351: Automatic control*

IEC 60050-351:2006, *International Electrotechnical Vocabulary – Part 351: Control technology*

IEC 61131-3, *Programmable controllers – Part 3: Programming languages*

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

IEC/TR 61804-4, *Function blocks (FB) for process control – Part 4: EDD interoperability guideline*

ISO 15745-1, *Industrial automation systems and integration – Open systems application integration framework – Part 1: Generic reference description*

ISO 2382 (all parts), *Information technology – Vocabulary*

ISO/AFNOR, *Dictionary of computer science – The standardized vocabulary*

Withd
IECNORM.COM: Click to view the full PDF of IEC 61804-3:2010

SOMMAIRE

AVANT-PROPOS	445
INTRODUCTION	448
1 Domaine d'application	449
2 Références normatives	450
3 Termes, définitions, termes abrégés et acronymes	450
3.1 Termes et définitions	450
3.2 Termes abrégés et acronymes	457
4 Déclaration de conformité	457
5 Conventions pour les structures lexicales	458
6 Modèle EDD et EDDL	459
6.1 Présentation des EDD et d'EDDL	459
6.2 Architecture EDD	459
6.3 Concepts d'EDD	460
6.4 Principes du processus de développement d'EDD	460
6.5 Interrelations entre la structure lexicale et les définitions formelles	462
6.6 Builtins	462
6.7 Profils	462
7 Langage de description de dispositif électronique	462
7.1 Vue d'ensemble	462
7.2 Informations d'identification d'une EDD	472
7.3 AXIS	475
7.4 BLOCK	477
7.5 CHART	489
7.6 COLLECTION	491
7.7 COMMAND	492
7.8 COMPONENT	499
7.9 COMPONENT_FOLDER	503
7.10 COMPONENT_REFERENCE	504
7.11 COMPONENT_RELATION	504
7.12 CONNECTION	507
7.13 DOMAIN	508
7.14 EDIT_DISPLAY	509
7.15 FILE	511
7.16 GRAPH	511
7.17 GRID	513
7.18 IMAGE	514
7.19 IMPORT	515
7.20 INTERFACE	531
7.21 LIKE	532
7.22 LIST	532
7.23 MENU	533
7.24 METHOD	541
7.25 PROGRAM	543
7.26 RECORD	544
7.27 REFERENCE_ARRAY	544
7.28 Relations	545

7.29 RESPONSE_CODES.....	547
7.30 SOURCE.....	547
7.31 TEMPLATE.....	549
7.32 VALUE_ARRAY.....	550
7.33 VARIABLE.....	551
7.34 VARIABLE_LIST.....	569
7.35 WAVEFORM.....	569
7.36 Attributs communs.....	576
7.37 Expression conditionnelle.....	584
7.38 Référencement.....	586
7.39 Chaînes.....	596
7.40 Expression.....	598
7.41 Dictionnaire de texte.....	603
Annexe A (normative) Définition de format EDDL.....	605
Annexe B (normative) Bibliothèque de Builtin EDDL.....	691
Annexe C (informative) Exemple d'EDD.....	863
Annexe D (normative) Profils d'EDDL et de Builtin.....	877
Annexe E (informative) Contexte historique.....	910
Bibliographie.....	912
Figure 1 – Position de la série CEI 61804 par rapport à d'autres normes et produits.....	448
Figure 2 – Processus de génération d'EDD.....	461
Figure 3 – BLOCK_A.....	464
Figure 4 – CHART.....	464
Figure 5 – COLLECTION.....	465
Figure 6 – COMMAND.....	465
Figure 7 – DOMAIN.....	465
Figure 8 – EDIT_DISPLAY.....	466
Figure 9 – FILE.....	466
Figure 10 – GRAPH.....	466
Figure 11 – GRID.....	466
Figure 12 – IMAGE.....	466
Figure 13 – LIKE.....	467
Figure 14 – LIST.....	467
Figure 15 – MENU.....	468
Figure 16 – PROGRAM.....	468
Figure 17 – RECORD.....	468
Figure 18 – REFERENCE_ARRAY.....	469
Figure 19 – REFRESH.....	469
Figure 20 – UNIT.....	469
Figure 21 – WRITE_AS_ONE.....	469
Figure 22 – SOURCE.....	470
Figure 23 – VALUE_ARRAY.....	470
Figure 24 – VARIABLE.....	470
Figure 25 – VARIABLE_LIST.....	470

Figure 26 – WAVEFORM	471
Figure 27 – Mécanismes d'importation EDDL	516
Figure 28 – Activation de MENU	541
Figure 29 – Délai d'attente d'opération de lecture/écriture	568
Figure C.1 – Exemple d'écran d'opérateur d'utilisation d'une EDD	863
Tableau 1 – Descriptions d'attribut de champ	458
Tableau 2 – Attribut de DD_REVISION	473
Tableau 3 – Attribut de DEVICE_REVISION	473
Tableau 4 – Attributs de DEVICE_TYPE	473
Tableau 5 – Attribut de EDD_PROFILE	474
Tableau 6 – Attribut de EDD_VERSION	474
Tableau 7 – Attributs de MANUFACTURER	474
Tableau 8 – Attribut de MANUFACTURER_EXT	475
Tableau 9 – Attributs d'AXIS	475
Tableau 10 – Attribut de MAX_VALUE, MIN_VALUE	476
Tableau 11 – Attribut de SCALING	477
Tableau 12 – Attributs de BLOCK_A	478
Tableau 13 – Attribut de CHARACTERISTICS	479
Tableau 14 – Attributs de PARAMETERS	479
Tableau 15 – Attribut d'AXIS_ITEMS	479
Tableau 16 – Attribut de CHART_ITEMS	480
Tableau 17 – Attribut de COLLECTION_ITEMS	480
Tableau 18 – Attribut d'EDIT_DISPLAY_ITEMS	480
Tableau 19 – Attribut de FILE_ITEMS	481
Tableau 20 – Attribut de GRAPH_ITEMS	481
Tableau 21 – Attribut de GRID_ITEMS	481
Tableau 22 – Attribut d'IMAGE_ITEMS	481
Tableau 23 – Attribut de LIST_ITEMS	482
Tableau 24 – Attribut de MENU_ITEMS	482
Tableau 25 – Attribut de METHOD_ITEMS	483
Tableau 26 – Attributs de PARAMETER_LISTS	483
Tableau 27 – Attribut de REFERENCE_ARRAY_ITEMS	483
Tableau 28 – Attribut de REFRESH_ITEMS	484
Tableau 29 – Attribut de SOURCE_ITEMS	484
Tableau 30 – Attribut de UNIT_ITEMS	484
Tableau 31 – Attribut de WAVEFORM_ITEMS	484
Tableau 32 – Attribut de WRITE_AS_ONE_ITEMS	485
Tableau 33 – Attribut de CHARTS	485
Tableau 34 – Attributs de LISTS	485
Tableau 35 – Attributs de GRAPHS	486
Tableau 36 – Attributs de GRIDS	486
Tableau 37 – Attributs de MENUS	487

Tableau 38 – Attributs de METHODS	487
Tableau 39 – Attributs de BLOCK_B	488
Tableau 40 – Attributs de NUMBER	488
Tableau 41 – Attributs de TYPE	489
Tableau 42 – Attributs de CHART	489
Tableau 43 – Attribut de CYCLE_TIME	490
Tableau 44 – Attribut de LENGTH	490
Tableau 45 – Attribut de TYPE	490
Tableau 46 – Attributs de COLLECTION	491
Tableau 47 – item-type	492
Tableau 48 – Attributs de COMMAND	493
Tableau 49 – Attributs d'OPERATION	493
Tableau 50 – Attributs de TRANSACTION	494
Tableau 51 – Attributs de REPLY et REQUEST	495
Tableau 52 – Attributs d'INDEX	496
Tableau 53 – Attribut de BLOCK_B	496
Tableau 54 – Attributs de NUMBER	497
Tableau 55 – Attributs de SLOT	497
Tableau 56 – Attributs de SUB_SLOT	498
Tableau 57 – Attribut de CONNECTION	498
Tableau 58 – Attribut de HEADER	498
Tableau 59 – Attribut de MODULE	499
Tableau 60 – Attributs de COMPONENT	499
Tableau 61 – Attributs de CAN_DELETE	500
Tableau 62 – Attributs de CHECK_CONFIGURATION	500
Tableau 63 – Attribut de COMPONENT_RELATIONS	501
Tableau 64 – Attribut de DECLARATION	501
Tableau 65 – Attribut de DETECT	501
Tableau 66 – Attribut d'EDD	502
Tableau 67 – Attributs de CHECK_CONFIGURATION	502
Tableau 68 – Attribut de REDUNDANCY	502
Tableau 69 – Attribut de SCAN	503
Tableau 70 – Attribut de SCAN_LIST	503
Tableau 71 – Attributs de COMPONENT_FOLDER	503
Tableau 72 – Attribut de COMPONENT_REFERENCE	504
Tableau 73 – Attributs de COMPONENT_RELATION	505
Tableau 74 – Attributs de COMPONENTS	505
Tableau 75 – Attributs de RELATION_TYPE	506
Tableau 76 – Attribut d'ADDRESSING	506
Tableau 77 – Attribut de MAXIMUM_NUMBER	507
Tableau 78 – Attribut de MINIMUM_NUMBER	507
Tableau 79 – Attribut de REQUIRED_INTERFACE	507
Tableau 80 – Attribut de CONNECTION	508

Tableau 81 – Attributs d'APPINSTANCE.....	508
Tableau 82 – Attributs de DOMAIN	508
Tableau 83 – Attributs de EDIT_DISPLAY.....	509
Tableau 84 – Attribut d'EDIT_ITEMS	509
Tableau 85 – Attribut de DISPLAY_ITEM	510
Tableau 86 – Attributs de POST_EDIT_ACTIONS, PRE_EDIT_ACTIONS	511
Tableau 87 – Attributs de FILE	511
Tableau 88 – Attributs de GRAPH.....	512
Tableau 89 – Attribut de CYCLE_TIME	512
Tableau 90 – Attribut de X_AXIS	513
Tableau 91 – Attributs de GRID	513
Tableau 92 – Attribut de VECTORS	514
Tableau 93 – Attribut d'ORIENTATION	514
Tableau 94 – Attributs d'IMAGE.....	514
Tableau 95 – Attribut de PATH	515
Tableau 96 – Attribut de LINK.....	515
Tableau 97 – Importation de description d'un dispositif.....	517
Tableau 98 – Attributs de redéfinition.....	518
Tableau 99 – Règles de redéfinition pour les attributs d'AXIS	518
Tableau 100 – Règles de redéfinition pour les attributs de BLOCK_A	519
Tableau 101 – Règles de redéfinition pour les attributs de BLOCK_B	520
Tableau 102 – Règles de redéfinition pour les attributs de CHART	520
Tableau 103 – Règles de redéfinition pour les attributs de COLLECTION	521
Tableau 104 – Règles de redéfinition pour les attributs de COMMAND	521
Tableau 105 – Règles de redéfinition pour les attributs de COMPONENT	521
Tableau 106 – Règles de redéfinition pour les attributs de COMPONENT_FOLDER	522
Tableau 107 – Règles de redéfinition pour les attributs de COMPONENT_REFERENCE	522
Tableau 108 – Règles de redéfinition pour les attributs de COMPONENT_RELATION	523
Tableau 109 – Règles de redéfinition pour les attributs de CONNECTION	523
Tableau 110 – Règles de redéfinition pour les attributs de DOMAIN	523
Tableau 111 – Règles de redéfinition pour les attributs d'EDIT_DISPLAY	524
Tableau 112 – Règles de redéfinition pour les attributs de FILE.....	524
Tableau 113 – Règles de redéfinition pour les attributs de GRAPH	524
Tableau 114 – Règles de redéfinition pour les attributs de GRID	525
Tableau 115 – Règles de redéfinition pour les attributs d'IMAGE	525
Tableau 116 – Règles de redéfinition pour les attributs d'INTERFACE.....	525
Tableau 117 – Règles de redéfinition pour les attributs de LIST	525
Tableau 118 – Règles de redéfinition pour les attributs de MENU	526
Tableau 119 – Règles de redéfinition pour les attributs de METHOD	526
Tableau 120 – Règles de redéfinition pour les attributs de PROGRAM	527
Tableau 121 – Règles de redéfinition pour les attributs de RECORD	527
Tableau 122 – Règles de redéfinition pour les attributs de REFERENCE_ARRAY	527

Tableau 123 – Règles de redéfinition pour les attributs de RESPONSE_CODES	528
Tableau 124 – Règles de redéfinition pour les attributs de SOURCE	528
Tableau 125 – Règles de redéfinition pour les attributs de TEMPLATE	528
Tableau 126 – Règles de redéfinition pour les attributs de VALUE_ARRAY	529
Tableau 127 – Règles de redéfinition pour les attributs de VARIABLE	529
Tableau 128 – Règles de redéfinition pour les attributs de VARIABLE_LIST	530
Tableau 129 – Règles de redéfinition pour les attributs de WAVEFORM	530
Tableau 130 – Attributs d'INTERFACE	531
Tableau 131 – Attributs de DECLARATION	531
Tableau 132 – Attributs de LIKE	532
Tableau 133 – Attributs de LIST	532
Tableau 134 – Attribut de TYPE	533
Tableau 135 – Attribut de CAPACITY, COUNT	533
Tableau 136 – Attributs de MENU	534
Tableau 137 – Attributs d'ITEMS	535
Tableau 138 – Attributs d'ACCESS	535
Tableau 139 – Attributs POST_EDIT_ACTIONS, PRE_EDIT_ACTIONS, POST_READ_ACTIONS, PRE_READ_ACTIONS, POST_WRITE_ACTIONS, PRE_WRITE_ACTIONS	536
Tableau 140 – Attributs de PURPOSE	538
Tableau 141 – Attribut de ROLE	539
Tableau 142 – Attributs de STYLE	540
Tableau 143 – Attributs de METHOD	542
Tableau 144 – Attributs d'ACCESS	542
Tableau 145 – Attributs de TYPE	543
Tableau 146 – Attributs de PROGRAM	543
Tableau 147 – Attributs d'ARGUMENTS	544
Tableau 148 – Attributs de RECORD	544
Tableau 149 – Attributs de REFERENCE_ARRAY	545
Tableau 150 – Attributs d'ELEMENTS	545
Tableau 151 – Attributs de REFRESH	546
Tableau 152 – Attributs d'UNIT	546
Tableau 153 – Attribut de WRITE_AS_ONE	546
Tableau 154 – Attributs de RESPONSE_CODES	547
Tableau 155 – Attributs de SOURCE	547
Tableau 156 – Attribut de Y_AXIS	549
Tableau 157 – Attributs de TEMPLATE	550
Tableau 158 – Attributs de DEFAULT_VALUES	550
Tableau 159 – Attributs de VALUE_ARRAY	550
Tableau 160 – Attributs de NUMBER_OF_ELEMENTS	551
Tableau 161 – Attribut de TYPE	551
Tableau 162 – Attributs de VARIABLE	552
Tableau 163 – Attributs de CLASS	553
Tableau 164 – Attributs de TYPE	554

Tableau 165 – Attributs de DOUBLE, FLOAT, INTEGER, UNSIGNED_INTEGER.....	555
Tableau 166 – Attributs de DATE, DATE_AND_TIME, DURATION, TIME, TIME_VALUE.....	558
Tableau 167 – Attributs de BIT_ENUMERATED.....	559
Tableau 168 – Attributs de status-class.....	560
Tableau 169 – Attributs ALL, AO, DV, TV.....	561
Tableau 170 – Attributs de types énumérés.....	561
Tableau 171 – Attributs de type index.....	562
Tableau 172 – Attribut du type de référence d'objet.....	562
Tableau 173 – Attributs de DEFAULT_REFERENCE.....	563
Tableau 174 – Attributs de types de chaîne.....	564
Tableau 175 – Attribut CONSTANT_UNIT.....	565
Tableau 176 – Attribut DEFAULT_VALUE.....	565
Tableau 177 – Attribut d'INITIAL_VALUE.....	565
Tableau 178 – Attributs de POST_EDIT_ACTIONS, PRE_EDIT_ACTIONS, POST_READ_ACTIONS, PRE_READ_ACTIONS, POST_WRITE_ACTIONS, PRE_WRITE_ACTIONS, REFRESH_ACTIONS.....	566
Tableau 179 – Attributs de READ/WRITE_TIMEOUT.....	568
Tableau 180 – Attribut de STYLE.....	569
Tableau 181 – Attributs de VARIABLE_LIST.....	569
Tableau 182 – Attributs de WAVEFORM.....	570
Tableau 183 – Attribut de TYPE.....	570
Tableau 184 – Attribut XY.....	571
Tableau 185 – Attributs de YT.....	571
Tableau 186 – Attribut d'HORIZONTAL.....	572
Tableau 187 – Attribut de VERTICAL.....	573
Tableau 188 – Attributs d'EXIT_ACTIONS, INIT_ACTIONS, REFRESH_ACTIONS.....	574
Tableau 189 – Attributs de KEY_POINTS.....	574
Tableau 190 – Attribut de X_VALUES, Y_VALUES.....	574
Tableau 191 – Attribut de Y_AXIS.....	575
Tableau 192 – Attribut de CLASSIFICATION.....	576
Tableau 193 – Attribut de COMPONENT_PARENT.....	578
Tableau 194 – Attribut de COMPONENT_PATH.....	578
Tableau 195 – Attribut de DEFINITION.....	579
Tableau 196 – Attribut d'EMPHASIS.....	579
Tableau 197 – Attributs de HANDLING.....	579
Tableau 198 – Attributs de HEIGHT/WIDTH.....	580
Tableau 199 – Attribut de HELP.....	580
Tableau 200 – Attribut de LABEL.....	581
Tableau 201 – Attributs de LINE_COLOR.....	581
Tableau 202 – Attribut de LINE_TYPE.....	581
Tableau 203 – Attributs de MEMBERS.....	582
Tableau 204 – Attribut de PROTOCOL.....	583
Tableau 205 – Attribut de RESPONSE_CODES.....	583

Tableau 206 – Attribut de SUPPLIED_INTERFACE	584
Tableau 207 – Attributs de VALIDITY	584
Tableau 208 – Instruction conditionnelle IF, SELECT	585
Tableau 209 – Référencement d'une instance d'EDD	586
Tableau 210 – Référencement d'éléments de VARIABLE	586
Tableau 211 – Référencement d'éléments de RECORD	587
Tableau 212 – Référencement d'éléments de VALUE_ARRAY	587
Tableau 213 – Référencement des membres de COLLECTION	587
Tableau 214 – Référencement des membres de REFERENCE_ARRAY	588
Tableau 215 – Référencement des membres de VARIABLE_LISTS	588
Tableau 216 – Référencement des membres de BLOCK_A PARAMETERS	588
Tableau 217 – Référencement des membres de BLOCK_A PARAMETER_LISTS	589
Tableau 218 – Référencement des membres de BLOCK_A LOCAL_PARAMETERS	589
Tableau 219 – Référencement de BLOCK_A CHARACTERISTICS	589
Tableau 220 – Référencement des membres de FILE	589
Tableau 221 – Référencement d'éléments de LIST	590
Tableau 222 – Référencement des membres de CHART	590
Tableau 223 – Référencement des membres de GRAPH	590
Tableau 224 – Référencement des membres de SOURCE	591
Tableau 225 – Référencement d'AXIS de GRAPH, SOURCE, WAVEFORM	591
Tableau 226 – Référencement de PARAMETERS d'une instance spécifique de BLOCK_A	592
Tableau 227 – Référencement de LOCAL_PARAMETERS d'une instance spécifique de BLOCK_A	592
Tableau 228 – Référencement de CHARACTERISTICS d'une instance spécifique de BLOCK_A	592
Tableau 229 – Référencement de CHARTS d'une instance spécifique de BLOCK_A	593
Tableau 230 – Référencement de LISTS d'une instance spécifique de BLOCK_A	593
Tableau 231 – Référencement de GRAPHS d'une instance spécifique de BLOCK_A	593
Tableau 232 – Référencement de GRIDS d'une instance spécifique de BLOCK_A	594
Tableau 233 – Référencement de MENUS d'une instance spécifique de BLOCK_A	594
Tableau 234 – Référencement de METHODS d'une instance spécifique de BLOCK_A	595
Tableau 235 – Référencement d'une instance de COMPONENT	595
Tableau 236 – Référencement d'un type COMPONENT	595
Tableau 237 – Chaîne sous forme de littéral chaîne	596
Tableau 238 – Chaîne en tant que variable chaîne	596
Tableau 239 – Chaîne en tant que valeur d'énumération	596
Tableau 240 – Chaîne en tant que référence de dictionnaire	597
Tableau 241 – Référencement des attributs HELP et LABEL des instances de l'EDD	597
Tableau 242 – Opération de chaîne	598
Tableau 243 – Spécificateur de format	598
Tableau 244 – Expressions primaires	599
Tableau 245 – Valeurs d'attribut de VARIABLE	599
Tableau 246 – Valeurs d'attribut d'AXIS	600

Tableau 247 – Valeurs d’attribut de LIST	600
Tableau 248 – Expressions unaires	600
Tableau 249 – Opérateurs multiplicatifs	601
Tableau 250 – Opérateurs additifs	601
Tableau 251 – Opérateurs de décalage	601
Tableau 252 – Opérateurs relationnels	602
Tableau 253 – Opérateurs d’égalité	602
Tableau 254 – Attributs de dictionnaire de texte	603
Tableau A.1 – Conventions pour les constantes entières	609
Tableau A.2 – Utilisation de séquences d’échappement dans des chaînes littérales	610
Tableau A.3 – Utilisation de codes de langue dans des chaînes littérales	610
Tableau A.4 – Opérateurs EDDL	611
Tableau A.5 – Mots clés EDDL	611

IECNORM.COM : Click to view the full PDF of IEC 61804-3:2010

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

BLOCS FONCTIONNELS (FB) POUR LES PROCÉDÉS INDUSTRIELS –**Partie 3: Langage de description électronique de produit (EDDL)****AVANT-PROPOS**

- 1) La Commission Electrotechnique Internationale (CEI) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de la CEI). La CEI a pour objet de favoriser la coopération internationale pour toutes les questions de normalisation dans les domaines de l'électricité et de l'électronique. A cet effet, la CEI – entre autres activités – publie des Normes internationales, des Spécifications techniques, des Rapports techniques, des Spécifications accessibles au public (PAS) et des Guides (ci-après dénommés "Publication(s) de la CEI"). 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 la CEI, participent également aux travaux. La CEI 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 la CEI concernant les questions techniques représentent, dans la mesure du possible, un accord international sur les sujets étudiés, étant donné que les Comités nationaux de la CEI intéressés sont représentés dans chaque comité d'études.
- 3) Les Publications de la CEI se présentent sous la forme de recommandations internationales et sont agréées comme telles par les Comités nationaux de la CEI. Tous les efforts raisonnables sont entrepris afin que la CEI s'assure de l'exactitude du contenu technique de ses publications; la CEI 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 la CEI s'engagent, dans toute la mesure possible, à appliquer de façon transparente les Publications de la CEI dans leurs publications nationales et régionales. Toutes divergences entre toutes Publications de la CEI et toutes publications nationales ou régionales correspondantes doivent être indiquées en termes clairs dans ces dernières.
- 5) La CEI 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 la CEI. La CEI 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 la CEI, 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 la CEI, 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 la CEI ou de toute autre Publication de la CEI, 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 la CEI peuvent faire l'objet de droits de brevet. La CEI 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 CEI 61804-3 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 la CEI: Mesure, commande et automation dans les processus industriels.

Cette deuxième édition annule et remplace la première édition parue en 2006. Cette édition constitue une révision technique.

Les principales modifications par rapport à l'édition précédente sont les suivantes¹.

Ajouts

- Éléments de langage pour décrire les dispositifs modulaires, 7.9, 7.10, 7.36.1, 7.36.2, 7.36.3, 7.36.13 et 7.36.15;
- Éléments de langage pour décrire la configuration hors connexion, 7.31;

¹ Le contexte historique de la spécification EDDL est présenté à l'Annexe E.

- Utilisation de UTF-8, voir A.2.3;
 - Divers BuiltIn;
 - Référencement de BLOCK_A. Mises à jour pour décrire l'accès à des blocs multiples, voir 7.4.1, 7.38.18 à 7.38.26;
 - Ajout de l'attribut VALIDITY (validité) à différentes structures lexicales;
 - Prise en charge d'images multilingues, voir 7.18.21;
 - En 7.23.1, suppression de la restriction de la liste d'éléments de MENU;
 - Limitation syntaxique sur les instructions conditionnelles en 7.23.2.1 pour restreindre la disposition de MENU;
 - Ajout de LIST (liste) et suppression de VARIABLE_LIST (liste des variables) dans les éléments de MENU en 7.23.2.1;
 - Types de données de valeur de retour additionnels pour METHOD_TYPE, voir 7.24.2.3;
 - Remplacement de la référence par une spécification spécifique au contexte dans le Tableau 170;
 - Clarification sur le comportement des KEY_POINT (point clé), voir 7.35.2.4;
 - Ajout de TRANSPARENT à la structure lexicale pour rendre la liste d'attributs cohérente, voir 7.36.11;
 - Clarification sur le comportement des fichiers, voir 7.36.10;
 - Ajout dans le Tableau 5 de nouveaux mots clés;
 - Ajout de la syntaxe EDDL formelle en A.6.
- Corrections
 - Suppression en A.6 de toutes les constructions non nécessaires utilisant l'élément auxiliaire `...listR`. Celles-ci avaient été créées par un outil de vérification de syntaxe imparfait.
 - Suppression de tous les `"stmt1:"` et `"stmt2:"` superflus en A.6. Ceux-ci avaient été créés par un outil de vérification de syntaxe imparfait.
 - A.6, désormais cohérent en ce qui concerne l'utilisation d'un deux-points à la fin d'un terme après modification des deux-points.
 - Références non utilisées supprimées.
 - Parenthèses et `"<exp>"` supprimés dans plusieurs structures lexicales.
 - `"[... <exp>]"` supprimé dans plusieurs structures lexicales dans lesquelles l'expression conditionnelle n'est pas possible.
 - Attribut WIDTH (largeur) remplacé par HEIGHT (hauteur) dans le paragraphe spécifiant HEIGHT.
 - Fautes d'orthographe telles que GUAGE remplacé par GAUGE.
 - Limitation syntaxique sur les instructions conditionnelles dans les ACTIONS (par exemple, voir 7.14.2.3 et 7.14.2.4) pour prendre en charge les éléments conditionnels uniquement dans les éléments METHOD (méthode).
 - Clarification sur le comportement des fichiers en 7.15.
 - Ajout de GRID (grille) et IMAGES à la liste des attributs, voir 7.19.
 - Suppression d'une duplication de liste d'éléments en 7.23.2.1.
 - Définition et appel des METHOD avec un paramètre et une valeur de retour, voir 7.24.
 - Restriction sur METHOD CLASS (classe de méthode), voir 7.24.2.2.

- Clarification sur le comportement de SCALING_FACTOR (facteur d'échelle), voir Tableau 165.
- Suppression d'opérateurs EDDL de la liste de mots-clés EDDL, voir Tableau A.5.
- Clarification sur le codage de TIME_VALUE (valeur d'heure) constituant la base absolue, voir Tableau D.16 et Tableau D.17.

La présente version bilingue, publiée en 2011-07, correspond à la version anglaise.

Le texte anglais de cette norme est issu des documents 65E/162/FDIS et 65E/173/RVD.

Le rapport de vote 65E/173/RVD donne toute information sur le vote ayant abouti à l'approbation de cette norme.

La version française de cette norme n'a pas été soumise au vote.

Cette publication a été rédigée selon les Directives ISO/CEI, Partie 2.

Une liste de toutes les parties de la série CEI 61804, publiées sous le titre *Blocs fonctionnels (FB) pour les procédés industriels*, peut être consultée sur le site web de la CEI.

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 la CEI sous "<http://webstore.iec.ch>" dans les données relatives à la publication recherchée. A cette date, la publication sera

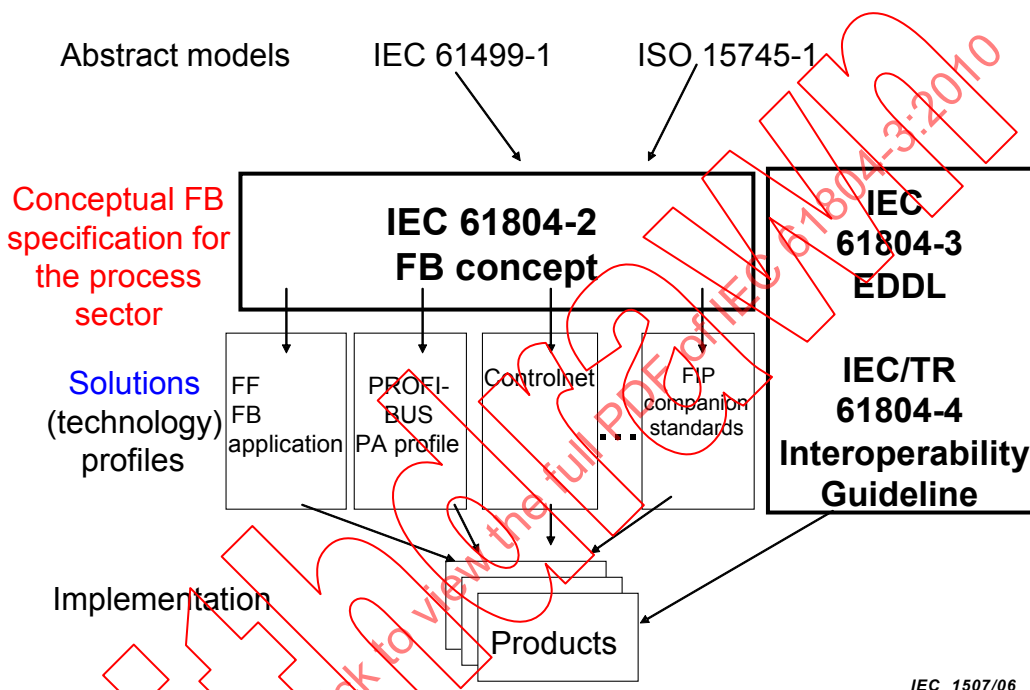
- reconduite,
- supprimée,
- remplacée par une édition révisée, ou
- amendée.

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 la série CEI 61804 spécifie la technologie de langage de description électronique de produit (EDDL).

Le langage EDDL établit un lien entre la spécification conceptuelle de bloc fonction de la CEI 61804-2 et une implémentation de produit. Il permet aux fabricants d'utiliser la même méthode de description pour des dispositifs basés sur différentes technologies et diverses plates-formes. La Figure 1 présente ces aspects.



Légende

Anglais	Français
Abstract models	Modèles abstraits
Conceptual FB specification for the process sector	Spécification conceptuelle de FB pour le secteur de processus
Solutions (technology) profiles	Profils (technologie) de solution
FF FB application	FF FB application
PROFIBUS PA profile	profil PA PROFIBUS
FIP companion standards	normes apparentées FIP
FB concept	Concept de FB
Implementation	Implémentation
Products	Produits
IEC 61804-3 EDDL	CEI 61804-3 EDDL
IEC/TR 61804-4 Interoperability Guideline	CEI/TR 61804-4 Directive d'interopérabilité
IEC 61499-1	CEI 61499-1

Figure 1 – Position de la série CEI 61804 par rapport à d'autres normes et produits

BLOCS FONCTIONNELS (FB) POUR LES PROCÉDÉS INDUSTRIELS –

Partie 3: Langage de description électronique de produit (EDDL)

1 Domaine d'application

La présente partie de la série CEI 61804 spécifie la technologie de langage de description électronique de produit (*Electronic Device Description Language*: EDDL), qui permet en utilisant les outils d'ingénierie l'intégration de produit dans les systèmes tout au long du cycle de vie.

La présente norme spécifie EDDL en tant que langage générique pour décrire les propriétés des composants système mettant en œuvre des automatismes. EDDL est capable de décrire

- les paramètres des dispositifs et leurs dépendances;
- les fonctions des dispositifs, par exemple le mode de simulation, l'étalonnage;
- les représentations graphiques, par exemple les menus;
- les interactions avec les dispositifs de commande;
- les représentations graphiques:
 - interface utilisateur avancée;
 - système graphique;
- le repertoire des données mémorisées.

EDDL est destiné à être utilisé pour créer une description d'un dispositif électronique (*Electronic Device Description*: EDD). Cette EDD est utilisée par des outils appropriés pour générer un code interprété qui prend en charge la manipulation des paramètres, le fonctionnement et la surveillance des composants système mettant en œuvre des automatismes, tels que des E/S, des contrôleurs, des capteurs et des contrôleurs programmables. Les outils d'implémentation ne font pas partie du domaine d'application de la présente norme.

La présente norme spécifie la sémantique et la structure lexicale indépendamment de la syntaxe. Une syntaxe spécifique est définie à l'Annexe A mais il est possible d'utiliser le modèle sémantique avec d'autres syntaxes.

NOTE 1 Le langage EDDL peut également être utilisé pour la description des propriétés de produit pour d'autres domaines.

Le langage EDDL et l'EDD associée à un dispositif est applicable à l'automatisation industrielle.

NOTE 2 L'automatisation industrielle peut comprendre des dispositifs tels que des modules d'entrée/sortie numériques et analogiques génériques, des contrôleurs de mouvement, des interfaces homme-machine, des capteurs, des contrôleurs en boucle fermée, des codeurs, des vannes hydrauliques, et des contrôleurs programmables.

La présente Norme internationale satisfait aux exigences de l'Article 9 de la CEI/TS 61804-1.

2 Références normatives

Les documents de référence suivants sont indispensables pour l'application du présent document. 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).

CEI 61499-1:2005, *Function blocks – Part 1: Architecture* (disponible en anglais uniquement)

CEI/TS 61804-1:2003, *Function blocks (FB) for process control – Part 1: Overview of system aspects* (disponible en anglais uniquement)

ISO/CEI 2022, *Information technology – Character code structure and extension techniques* (disponible en anglais uniquement)

ISO/CEI 2375:2003, *Information technology – Procedure for registration of escape sequences and coded character sets* (disponible en anglais uniquement)

ISO/CEI 7498-1, *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Modèle de référence de base: Le modèle de base Information technology*

ISO/CEI 8859-1:1998, *Information technology – 8-bit single-byte coded graphic character sets – Part 1: Latin alphabet No. 1*

ISO/CEI 9899, *Programming languages – C* (disponible en anglais uniquement)

ISO/IEC 10646-1:2000, *Technologies de l'information – Jeu universel de caractères codés sur plusieurs octets (JUC) – Partie 1: Architecture et plan multilingue de base*

ISO 639, *Codes pour la représentation des noms de langue*

ISO 3166-1, 2006, *Codes pour la représentation des noms de pays et de leurs subdivisions – Partie 1: Codes de pays*

IEEE 754:1985 (R1990), *Binary Floating-Point Arithmetic*

RFC 3629:2003, *UTF-8, User Datagram Protocol*, disponible à l'adresse
<<http://www.ietf.org/rfc/rfc0768.txt>>

3 Termes, définitions, termes abrégés et acronymes

3.1 Termes et définitions

Pour les besoins du présent document, les termes et définitions suivants s'appliquent, dont certains ont été compilés à partir des documents référencés.

NOTE Les termes dans la CEI 60050-351 s'appliquent partiellement.

3.1.1

algorithme (algorithm)

ensemble fini de règles bien définies pour la solution d'un problème en un nombre fini d'opérations

3.1.2**attribut** (attribut)

description d'une caractéristique ou un aspect visible de l'extérieur d'un objet

NOTE Les attributs d'un objet contiennent des informations pour des parties variables d'un objet. Ils donnent typiquement des informations de statut ou régissent le fonctionnement d'un objet. Les attributs peuvent également affecter le comportement d'un objet. Les attributs sont divisés en attributs de classe et en attributs d'instance.

3.1.3**Built-in** (Built-in)

sous-programme prédéfini pour la communication et l'affichage exécuté par l'application EDD

3.1.4**configuration** (d'un système ou d'un dispositif)

étape dans une conception de système: sélection d'unités fonctionnelles, assignation de leurs emplacements et définition de leurs interconnexions

[CEI 61499-1]

3.1.5**données** (data)

représentation de faits, concepts ou instructions d'une manière formalisée adaptée pour une communication, une interprétation ou un traitement par des êtres humains ou par des moyens automatiques

[Dictionnaire de l'informatique ISO/AFNOR]

3.1.6**connexion de données** (data connection)

association établie entre des unités fonctionnelles pour le transport de données

[CEI 61499-1]

3.1.7**données d'entrée** (data input)

interface d'un FB qui reçoit des données depuis une connexion de données

[CEI 61499-1]

3.1.8**données de sortie** (data output)

interface d'un FB, qui transmet des données à une connexion de données

[CEI 61499-1]

3.1.9**type de données** (data type)

ensemble de valeurs conjointement avec un ensemble d'opérations autorisées

[série ISO 2382]

3.1.10**dispositif** (device)

entité physique indépendante capable d'exécuter une ou plusieurs fonctions dans un contexte particulier et délimitées par ses interfaces

[CEI 61499-1]

3.1.11

application EDD (EDD application)

programme utilisant l'EDD, ou une forme traduite quelconque, qui produit une fonctionnalité telle qu'une représentation de communication, une représentation de données, une représentation graphique, etc.

3.1.12

processeur EDDL (EDDL processor)

processeur ou programme, qui traduit l'EDD en forme exécutable qui peut être traitée par une application EDD

3.1.13

profil EDDL (EDDL profile)

sélection des éléments pris en charge de la structure lexicale EDDL comprenant les définitions de syntaxe pour une pluralité de consortiums spécifiques

3.1.14

langage de description de dispositif électronique (Electronic Device Description Language)

EDDL

méthodologie pour décrire un/des paramètre(s) d'un composant de système mettant en œuvre des automatismes

3.1.15

description de dispositif électronique (Electronic Device Description)

EDD

collection de données contenant le(s) paramètre(s) du dispositif, leurs dépendances, leur représentation graphique et une description des ensembles de données qui sont transférés.

NOTE La description du dispositif électronique est créée en utilisant le langage de description électronique de produit (EDDL).

3.1.16

source de description du dispositif électronique (Electronic Device Description Source)

EDDS

fichier ASCII contenant une description de dispositif spécifique

3.1.17

technologie de description de dispositif électronique (Electronic Device Description Technology)

EDDT

technologie qui comprend le processus de développement d'une EDD, l'usage d'une EDD et la chaîne d'outils impliquée

3.1.18

vérificateur de description de dispositif électronique (Electronic Device Description Checker)

outil de test qui vérifie la structure lexicale et partiellement la sémantique des sources d'EDD afin de garantir la conformité de sources d'EDD avec le langage EDD

3.1.19

compilateur de description de dispositif électronique (Electronic Device Description Compiler)

compilateur qui traduit la source d'EDD dans un format interne qui est utilisé par l'interpréteur d'EDD

3.1.20

interpréteur de description de dispositif électronique (Electronic Device Description Interpreter)

EDDI

interpréteur qui utilise la source d'EDD ou un format interne qui est donné par le compilateur EDDL pour fournir les informations d'EDD à l'utilisateur d'EDD

3.1.21

entité (entity)

élément particulier, tel qu'une personne, un emplacement, un processus, un objet, un concept, une association, ou un événement

[CEI 61499-1]

3.1.22

événement (event)

occurrence instantanée qui est significative pour planifier l'exécution d'un algorithme

[CEI 61499-1]

NOTE L'exécution d'un algorithme peut utiliser des variables associées à un événement.

3.1.23

exception (exception)

événement qui cause la suspension d'une exécution normale

[CEI 61499-1]

3.1.24

fonction (function)

fonction spécifique d'une entité ou son action caractéristique

[CEI 61499-1]

3.1.25

unité fonctionnelle (functional unit)

entité matérielle ou logicielle, ou les deux, capable de remplir une fonction spécifiée

[Dictionnaire de l'informatique ISO/AFNOR]

3.1.26

fonction bloc (function block) (instance de fonction block)

unité fonctionnelle logicielle comprenant une copie individuelle nommée d'une structure de données et des opérations associées spécifiées par un type de FB correspondant

[CEI 61499-1]

NOTE Des opérations typiques d'un FB comprennent une modification des valeurs des données dans sa structure de données associée.

3.1.27

matériel (hardware)

équipement physique, par opposition à des programmes, des procédures, des règles et documentation associée

[Dictionnaire de l'informatique ISO/AFNOR]

3.1.28

implémentation (implementation)

phase de développement dans laquelle le matériel et le logiciel d'un système deviennent opérationnels

[CEI 61499-1]

3.1.29

variable d'entrée (input variable)

variable dont la valeur est donnée par une entrée de données, et qui peut être utilisée dans une ou plusieurs opérations d'un FB

NOTE Un paramètre d'entrée d'un FB, comme défini dans la CEI 61131-3, est une variable d'entrée.

[CEI 61499-1]

3.1.30

instance (instance)

unité fonctionnelle comprenant une entité individuelle nommée avec les attributs d'un type défini

[CEI 61499-1]

3.1.31

nom d'instance (instance name)

identifiant associé à, et désignant, une instance

[CEI 61499-1]

3.1.32

instanciation (instantiation)

création d'une instance d'un type spécifié

[CEI 61499-1]

3.1.33

interface (interface)

frontière commune entre deux unités fonctionnelles, définie par des caractéristiques fonctionnelles, des caractéristiques de signal, ou d'autres caractéristiques appropriées

[CEI 60050-351:1998, 351-11-19, modifiée]

3.1.34

variable interne (internal variable)

variable dont la valeur est utilisée ou modifiée par une ou plusieurs opérations d'un FB mais n'est pas fournie par une entrée de données ou à une sortie de données

[CEI 61499-1]

3.1.35

invocation (invocation)

processus d'initiation de l'exécution de la séquence d'opérations spécifiée dans un algorithme

[CEI 61499-1]

3.1.36

mapping (mapping)

ensemble de valeurs ayant une correspondance définie avec les quantités ou valeurs d'un autre ensemble

[Dictionnaire de l'informatique ISO/AFNOR]

3.1.37**modèle** (model)

représentation d'un processus, dispositif ou concept du monde réel

[CEI 61499-1]

3.1.38**opération** (operation)

action bien définie qui, lorsqu'elle est appliquée à une combinaison admissible quelconque d'entités connues, produit une nouvelle entité

[Dictionnaire de l'informatique ISO/AFNOR]

3.1.39**variable de sortie** (output variable)

variable dont la valeur est établie par une ou plusieurs opérations d'un FB, et est transmise à une sortie de données

NOTE Un paramètre de sortie d'un FB, comme défini dans la CEI 61131-3, est une variable de sortie.

[CEI 61499-1]

3.1.40**paramètre** (parameter)

variable à laquelle est donnée une valeur constante pour une application spécifiée et qui peut désigner l'application

[Dictionnaire de l'informatique ISO/AFNOR]

3.1.41**préprocesseur** (preprocessor)

partie de l'environnement d'exécution qui transforme les informations fournies dans une EDD

3.1.42**directives de préprocesseur** (preprocessor directives)

description de conditions pour filtrer le code d'EDD avant compilation ou interprétation

NOTE Par exemple, une directive de préprocesseur permettant de définir des noms pour des constantes ou décrire des macros pour rendre le code plus facile à lire.

3.1.43**ressource** (resource)

unité fonctionnelle contenue dans un dispositif qui a un contrôle indépendant de son fonctionnement, et qui fournit différents services à des applications, comprenant la programmation et l'exécution d'algorithmes

NOTE 1 L'objet RESOURCE défini dans la CEI 61131-3 est un élément de langage de programmation correspondant à la ressource définie ci-dessus.

NOTE 2 Un dispositif contient une ou plusieurs ressources.

3.1.44**application de gestion de ressource** (resource management application)

application dont la fonction primaire est la gestion d'une ressource unique

[CEI 61499-1]

3.1.45

service (service)

capacité fonctionnelle d'une ressource, qui peut être modélisée par une séquence de primitives de service

[CEI 61499-1]

3.1.46

logiciel (software)

création intellectuelle comprenant les programmes, procédures, règles et documentation associée relatifs au fonctionnement d'un système

[CEI 61499-1]

3.1.47

système (system)

ensemble d'éléments corrélés considérés dans un contexte défini comme un ensemble et séparé de son environnement

[CEI 60050-351:VEI 351-21-20:2006, modifié]

NOTE 1 De tels éléments peuvent être à la fois des objets matériels et des concepts ainsi que les résultats de ceux-ci (par exemple, des formes d'organisation, des méthodes mathématiques et des langages de programmation).

NOTE 2 Le système est considéré comme étant séparé de l'environnement et d'autres systèmes externes par une surface imaginaire, qui peut couper les liens entre ceux-ci et le système en question.

3.1.48

dictionnaire de texte (text dictionary)

collection de textes multilingues ou autres dans l'EDD

NOTE Les références dans une EDD sont utilisées pour sélectionner un dictionnaire de texte approprié.

3.1.49

type (type)

élément logiciel, qui spécifie les attributs communs partagés par toutes les instances du type

[CEI 61499-1]

3.1.50

nom de type (type name)

identifiant associé à, et désignant, un type

[CEI 61499-1]

3.1.51

variable (variable)

entité logicielle qui peut prendre différentes valeurs, une à la fois

NOTE 1 Les valeurs d'une variable sont généralement restreintes à un certain type de données.

NOTE 2 Les variables sont décrites comme étant des variables d'entrée, des variables de sortie, et des variables internes.

[CEI 61499-1]

3.2 Termes abrégés et acronymes

ANSI	Institut national américain de normalisation
ANSI C	Institut national américain de normalisation pour le langage de programmation C (voir ISO/CEI 9899)
ASCII	Code standard américain pour l'échange d'informations (voir ISO/IEC 10646-1)
ASN.1	Notation de structure lexicale abstraite 1
CP	Profil de communication
CPF	Famille de profils de communication
DD	Description de dispositif
EDD	Description de dispositif électronique
EDDI	Interpréteur de description de dispositif électronique
EDDL	Langage de description électronique de produit
EUC	Code d'unité étendu (voir ISO/CEI 2022)
FB	Bloc fonctionnel
FIFO	Premier entré premier sorti
HCF	HART® ² Communication Foundation
IHM	Interface homme machine
HTML	Hypertext Mark-up Language
E/S	Entrée/Sortie
ID	Identifiant
OSI	Interconnexion de systèmes ouverts
PI	PROFIBUS & PROFINET ³ International
PNO	PROFIBUS Nutzerorganisation
wao	Write as one

4 Déclaration de conformité

Une déclaration de conformité à un profil de la présente Norme internationale doit être établie⁴ comme étant

- conforme à la CEI 61804-3:2010 Annexe D n <Type>

² HART® est une marque déposée du consortium HART Communication Foundation (HCF) (organisation à but non lucratif). Cette information est donnée à l'intention des utilisateurs de la présente norme et ne signifie nullement que la CEI approuve ou recommande le détenteur de l'appellation commerciale ou l'un quelconque de ses produits. La conformité à ce profil ne nécessite pas l'utilisation de l'appellation commerciale. L'utilisation de l'appellation commerciale nécessite l'autorisation du propriétaire de la marque.

³ PROFIBUS et PROFINET sont des appellations commerciales de PROFIBUS Nutzerorganisation e.V. (PNO), filiale de PROFIBUS & PROFINET International. PNO est une organisation à but non lucratif pour le support des technologies de bus de terrain PROFIBUS et PROFINET. Cette information est donnée à l'intention des utilisateurs de la présente norme et ne signifie nullement que la CEI approuve ou recommande le détenteur de l'appellation commerciale ou l'un quelconque de ses produits. La conformité à ce profil ne requiert pas l'utilisation de l'appellation commerciale. L'utilisation des appellations commerciales requiert l'autorisation du propriétaire du nom commercial.

⁴ Conformément aux directives ISO/CEI.

où le Type entre les crochets obliques < > est facultatif et les crochets obliques ne sont pas nécessairement inclus.

NOTE Le Type peut être, par exemple, le nom du protocole de communication utilisé tel que PROFIBUS, Foundation Fieldbus ou HART.

Les normes de produit ne doivent pas nécessairement comprendre des aspects d'évaluation de conformité (comprenant des dispositions QM), normatifs ou informatifs, autres que des dispositions pour l'essai de produit (évaluation et examen).

5 Conventions pour les structures lexicales

Le langage EDDL est généralement décrit à l'aide de structures lexicales dans lesquelles les éléments et la présence de champs sont spécifiés. Une forme générale des structures lexicales est présentée ci-dessous.

ABC champ1, champ2

ABC est un élément lexical. Cet élément doit être codé dans une syntaxe concrète. Il n'est pas nécessaire de coder cet élément avec le nom "ABC". Il est également possible de coder cet élément, par exemple, avec un numéro de balise (Tag).

Champ1 et champ2 sont des champs de l'élément lexical ABC. Chaque champ est obligatoire et peut avoir plus d'un attribut. Si un champ a des attributs, la présence des attributs est spécifiée dans une table. Une virgule sépare les champs 1 et 2. La virgule est un élément lexical et n'est pas explicitement codée.

Si un champ a des attributs additionnels, les attributs sont définis dans une table. La disposition de la table et les qualificatifs d'usage possibles sont présentés dans le Tableau 1.

Tableau 1 – Descriptions d'attribut de champ

Usage	Attribut	Description
m	yyy	La présence de cet attribut est obligatoire.
o	xxx	La présence de cet attribut est facultative.
s	z1	La présence de cet attribut est sélectionnable avec d'autres attributs, qui sont également marqués avec "s" dans la colonne d'usage). Un seul des attributs sélectionnables (z1 ou z2) est présent.
s	z2	La présence de cet attribut est sélectionnable avec d'autres attributs, qui sont également marqués avec "s" dans la colonne d'usage). Un seul des attributs sélectionnables (z1 ou z2) est présent.
c	uuu	La présence de cet attribut est conditionnelle et celui-ci est présent uniquement si la condition est vraie.

Les caractères dans la colonne Usage ont les définitions suivantes.

- m: cet attribut est obligatoire et doit être présent.
- o: cet attribut est facultatif et n'est pas nécessairement présent.
- s: cet attribut est une sélection. Un et un seul des champs marqués avec "s" (Z1 ou Z2) doit être présent.
- c: cet attribut est conditionnel. La condition est décrite dans la colonne de description.

Dans la colonne Attribut, s'il existe plusieurs attributs, qui ont le même usage, ceux-ci sont triés par ordre alphabétique.

ABC champ1+

Le signe plus (+) après champ1 est utilisé pour indiquer que champ1 est utilisé au moins une fois. Celui-ci peut être utilisé plus d'une fois.

ABC champ2*

L'astérisque (*) derrière champ2 est utilisé pour indiquer que champ2 est facultatif et n'est pas nécessairement utilisé. S'il est utilisé, il peut être utilisé plus d'une fois.

ABC [champ1, champ2]+

Les éléments champ1 et champ2 entre les crochets [] sont une liste non triée. Le signe plus derrière le crochet final indique que champ1 et champ2 sont utilisés au moins une fois et peuvent être utilisés plus d'une fois en tant que groupe.

ABC champ1, (champ2, champ3)<exp>

<exp> indique que champ2 et champ3 sont utilisés conjointement avec l'expression conditionnelle, (des constructions conditionnelles sont spécifiées en 7.37). L'expression <exp> désigne uniquement les champs entre parenthèses (). L'utilisation d'expressions conditionnelles est facultative.

6 Modèle EDD et EDDL**6.1 Présentation des EDD et d'EDDL**

Une description de dispositif électronique (EDD) contient tous le(s) paramètre(s) d'un dispositif d'un système d'automatismes. La technologie utilisée pour décrire une EDD est appelée langage de description électronique de produit (EDDL). Le langage EDDL comporte un ensemble d'éléments de langage basiques évolutifs pour gérer des dispositifs simples, complexes ou modulaires. Le langage EDDL est un langage descriptif basé sur un format ASCII avec une séparation nette entre les données et le programme.

NOTE Les données dans un champ de texte, qui sont marquées avec un code de pays, par exemple, japonais, peuvent utiliser un code multi-octets.

6.2 Architecture EDD

Du point de vue du modèle ISO/OSI (ISO/CEI 7498-1), une EDD est au-dessus de la Couche 7. Cependant, l'application EDD utilise le système de communication pour transférer ses informations. Une EDD contient des constructions qui prennent en charge le mapping du système de communication pris en charge.

Le fabricant du dispositif définit les objets, qui formeront la représentation logique des objets d'une application EDD. Pour cette raison, EDDL comprend des éléments de langage, qui mettent en correspondance les données de l'EDD avec la représentation de données du système de communication, de sorte que l'utilisateur d'une EDD n'a pas besoin de connaître l'emplacement physique (adresse) de l'objet dispositif (device object).

Une EDD décrit la gestion des informations à afficher à l'utilisateur. La représentation spécifique d'une telle visualisation ne fait pas partie des définitions d'EDD ou EDDL.

6.3 Concepts d'EDD

Le fabricant d'un dispositif ou d'un composant d'un système d'automatismes décrit les propriétés du dispositif en utilisant l'EDDL. L'EDD résultant contient des informations telles que:

- la description des paramètres du dispositif;
- la description des dépendances des paramètres;
- le regroupement logique des paramètres du dispositif;
- la sélection et l'exécution des fonctions du dispositif prises en charge;
- la description des ensembles des données transférées.

Suivant l'utilisation requise, l'EDD peut être physiquement située

- dans un dispositif;
- dans un support de stockage de données externe tel qu'un disque compact, une disquette ou un serveur;
- partiellement distribuée dans le dispositif et un support de stockage externe.

Une EDD prend en charge les chaînes de texte (termes communs, phrases, etc.) en plusieurs langues (anglais, allemand, français, etc.). Les chaînes de texte peuvent être stockées dans des dictionnaires séparés. Il peut y avoir plus d'un dictionnaire pour une EDD.

Une implémentation d'EDD comprend des informations nécessaires concernant le dispositif cible, par exemple, le fabricant, le type de dispositif, la révision, etc. Cela est utilisé pour mettre en correspondance une EDD spécifique avec un dispositif spécifique.

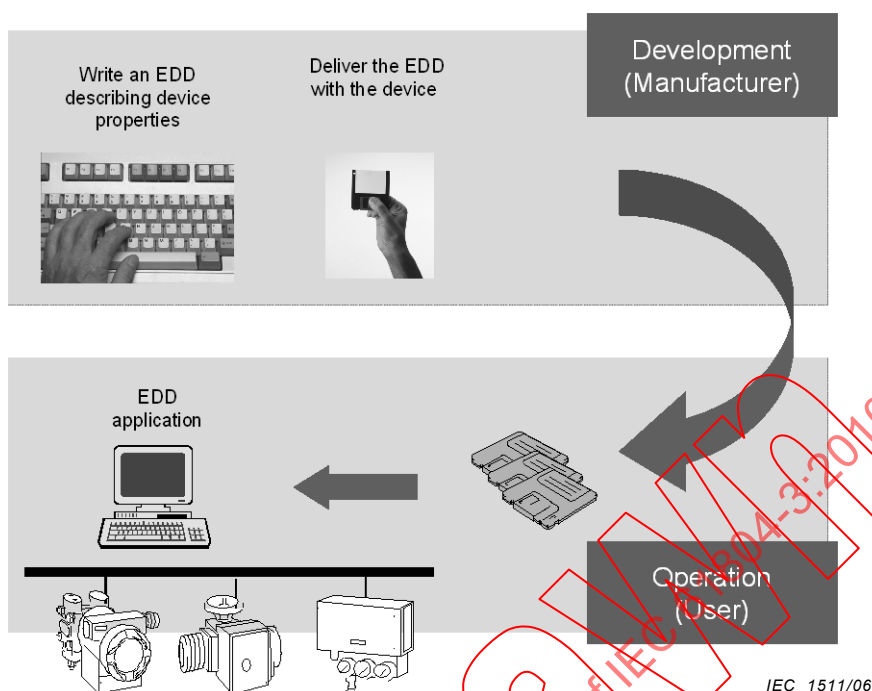
6.4 Principes du processus de développement d'EDD

6.4.1 Généralités

Le processus de création d'une EDD comporte trois étapes: génération du code source de l'EDD, prétraitement de l'EDD et compilation de l'EDD.

6.4.2 Génération de source d'EDD

Le processus de génération d'une EDD est représenté sur la Figure 2.



Légende

Anglais	Français
Write an EDD describing device properties	Écriture d'une EDD décrivant des propriétés du dispositif
Deliver the EDD with the device	Fourniture de l'EDD avec le dispositif
Development (manufacturer)	Développement (fabricant)
EDD application	Application de l'EDD
Operation (User)	Opération (utilisateur)

Figure 2 – Processus de génération d'EDD

Le fabricant d'un dispositif écrit une EDD appropriée pour son dispositif et fournit les deux (l'EDD et le dispositif). Si le système mettant en œuvre des automatismes prend en charge la méthode EDD, une intégration de système peut être effectuée par l'utilisateur.

Les EDD pour les dispositifs peuvent être embarquées dans la mémoire du dispositif ou fournis en utilisant des supports de stockage séparés ou téléchargés depuis un serveur réseau approprié. L'EDD est "interprétée" ou "parcourue" par une application EDD. Les EDD sont normalement stockées en tant que fichiers sources ou fichiers prétraités.

6.4.3 Prétraitement d'EDD

Dans l'étape de prétraitement, un préprocesseur d'EDD génère une représentation de l'EDD cohérente et adaptée pour une compilation finale.

Le prétraitement prend en charge, par exemple, la substitution des définitions et l'inclusion de texte externe. La sortie du préprocesseur est une EDD complète sans aucune directive de préprocesseur.

6.4.4 Compilation d'EDD

L'étape de compilation d'EDD produit une représentation interne d'application EDD à partir d'une EDD prétraitée destinée à être utilisée dans l'application EDD.

6.5 Interrelations entre la structure lexicale et les définitions formelles

La structure lexicale de l'EDDL et ses éléments sont décrits à l'Article 7. Les définitions et la syntaxe formelles pour chaque élément de l'EDDL sont décrites dans l'Annexe A. La structure lexicale et sa description formelle utilisent le même nom.

NOTE À la place des définitions et de la syntaxe formelles spécifiées dans l'Annexe A, une autre spécification peut être développée en tant qu'option additionnelle.

6.6 Builtins

Les Builtins sont des sous-programmes prédéfinis qui sont exécutés dans l'application EDD.

EXEMPLE Un terminal portatif est un dispositif simple ayant un petit écran et des fonctions de curseur limitées. Pour ce type de dispositif, un Builtin peut être spécifié pour produire une entrée d'affichage en utilisant uniquement des actions de curseur haut/bas gauche/droite.

La bibliothèque de Builtin est définie dans l'Annexe B.

6.7 Profils

EDDL est une spécification harmonisée de concepts EDD hérités existants. Les applications EDD concrètes utilisent un sous-ensemble de la spécification EDDL. La sélection d'éléments EDDL et de Builtin est effectuée à partir des profils définis dans l'Annexe D.

En plus des profils de l'EDD, les consortiums implémenteurs publient également des "profils de dispositif", qui sont utilisés pour prendre en charge l'interchangeabilité de dispositifs conformes. Ces profils de dispositifs peuvent être décrits en utilisant l'EDDL spécifié dans ce document.

7 Langage de description de dispositif électronique

7.1 Vue d'ensemble

7.1.1 Caractéristiques du langage EDDL

EDDL est un langage structuré et interprétatif pour décrire les propriétés d'un dispositif. De plus, les interactions entre les dispositifs et l'environnement d'exécution de l'EDD sont incorporés dans EDDL. Le langage EDDL comprend un ensemble d'éléments de langage pour ces applications.

Pour une implémentation d'EDD spécifique, il n'est pas nécessaire d'utiliser tous les éléments fournis par le langage.

Des sous-ensembles compatibles du langage EDDL sont autorisés et peuvent être spécifiés en utilisant des profils (par exemple, le choix de constructions, le nombre de récurrences, et la sélection d'options). Les profils pris en charge par certains consortiums industriels sont spécifiés dans l'Annexe D. Les développeurs d'EDD doivent identifier dans chaque dispositif les détails concernant le profil qui a été utilisé.

7.1.2 Représentation de syntaxe

La spécification du langage EDDL dans cet article de la norme utilise une syntaxe abstraite. La structure lexicale abstraite de cette section est convertie en syntaxes spécifiques spécifiées dans l'Annexe A par le nom de l'élément (par exemple, VARIABLE dans la structure lexicale équivaut au mot-clé "VARIABLE" dans l'annexe).

NOTE Hormis la syntaxe définie dans la présente norme, il est possible d'utiliser d'autres définitions de syntaxe, qui peuvent être ajoutées dans le futur pour permettre d'autres caractéristiques et représentations pour une évolution future.

7.1.3 Éléments de langage d'une EDD

Le langage est structuré dans les éléments de langage suivants:

- élément d'identification;
- éléments de construction de base;
- éléments spécifiques.

Il est recommandé que les chaînes de texte fréquemment utilisées, par exemple, un texte d'aide et des listes multilingues d'étiquettes soient séparées dans un dictionnaire de texte (voir 7.41).

Les informations d'identification doivent être les premières entrées dans chaque fichier EDD et doivent apparaître une seule fois. Les informations d'identification identifient uniquement la version du langage EDDL utilisée, conjointement avec le type de dispositif, les codes du modèle et les détails de révision spécifiques couverts par le fichier EDD.

7.1.4 Éléments de construction de base

Ces constructions de base ont été spécifiées pour prendre en charge les descriptions de dispositifs utilisés dans les applications de contrôle industriel, conjointement avec leurs propriétés et fonctionnalités.

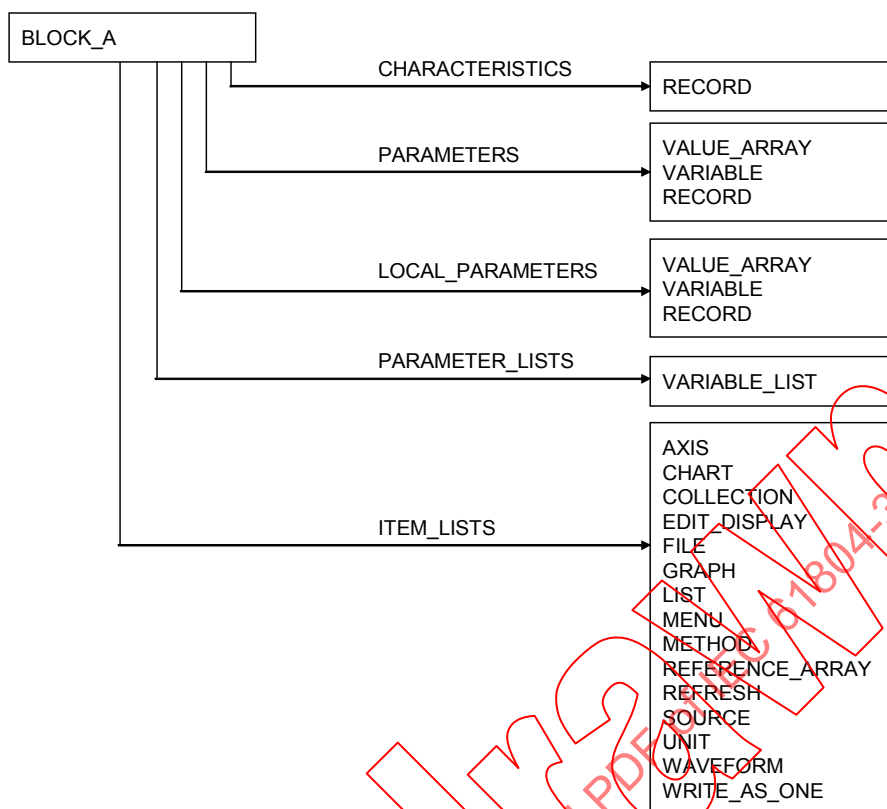
Certaines constructions ont des noms et fonctions similaires mais diffèrent dans leurs spécifications détaillées. Cette variété additionnelle a été incluse afin d'assurer la compatibilité avec plusieurs langages de description existants. Des références croisées de mapping appropriées sont données par les profils.

Chacune des constructions de base a un ensemble d'attributs associés à celle-ci. Les attributs peuvent avoir des sous-attributs, qui affinent la définition de l'attribut et par conséquent, la définition de la construction elle-même.

La définition d'un attribut peut être statique ou dynamique. Une définition d'attribut statique ne change jamais, tandis qu'une définition d'attribut dynamique peut changer afin de prendre en compte des changements de valeur de paramètre.

Certaines constructions de base font référence à d'autres constructions de base. Les constructions de base ainsi que les relations entre les constructions de base sont décrites dans la liste suivante:

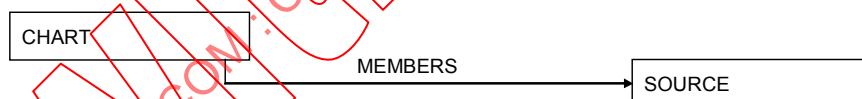
- **AXIS** décrit l'axe d'un CHART (diagramme) ou GRAPH (graphique) (voir 7.3).
- **BLOCK_A** est un regroupement logique de CHARACTERISTICS, PARAMETERS, PARAMETER_LISTS, et ITEM_LISTS, voir Figure 3. Pour accéder à un élément de BLOCK_A, il convient que l'instance du bloc soit utilisée (voir 7.4.1 et Figure 3).



IEC 1512/06

Figure 3 – BLOCK_A

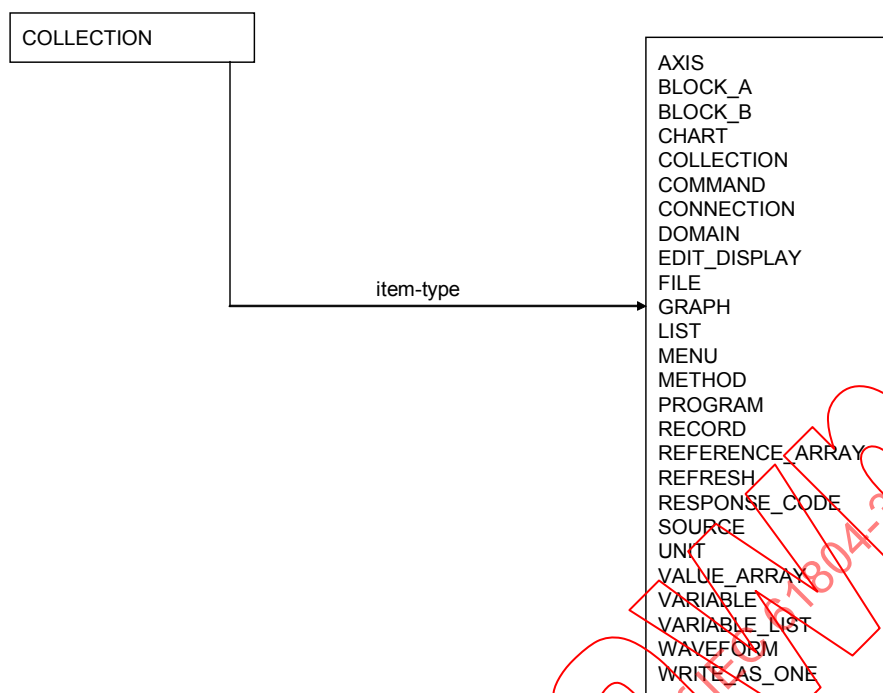
- **BLOCK_B** est utilisé pour structurer les variables en instances de types de bloc logique. Pour accéder à un VARIABLE dans une construction BLOCK_B, la construction de base COMMAND est utilisée pour générer un adressage relatif (voir 7.4.2).
- **CHART** (diagramme) décrit un diagramme utilisé pour afficher des données provenant d'un dispositif dans une application EDD (voir 7.5 et Figure 4).



IEC 1513/06

Figure 4 – CHART

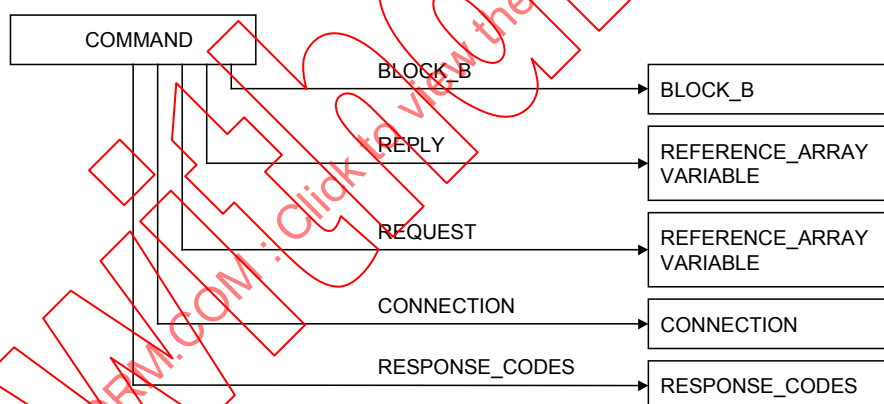
- **COLLECTION** décrit des regroupements logiques de données (voir 7.6 et Figure 5).



IEC 1514/06

Figure 5 – COLLECTION

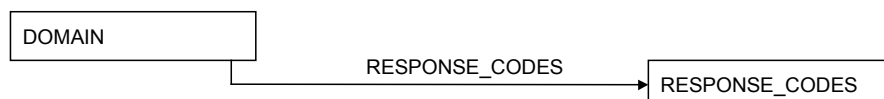
- **COMMAND** (commande) décrit la structure et l'adressage des variables dans le dispositif (voir 7.7 et Figure 6).



IEC 1515/06

Figure 6 – COMMAND

- **CONNECTION** (connexion) définit des applications multiples dans un dispositif (voir 7.8).
- **DOMAIN** (domaine) peut être utilisé pour télécharger ou transmettre des quantités modérément élevées de données vers ou depuis un dispositif (voir 7.13 et Figure 7).



IEC 1516/06

Figure 7 – DOMAIN

- **EDIT_DISPLAY** décrit comment les données seront présentées à un utilisateur par un dispositif d'affichage (voir 7.14 et Figure 8).

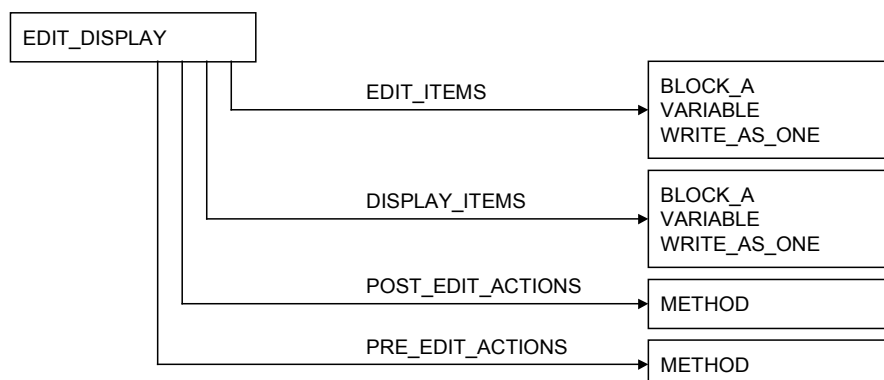


Figure 8 – EDIT_DISPLAY

IEC 1517/06

- **FILE** (fichier) décrit une zone de stockage persistant (voir 7.15 et Figure 9).

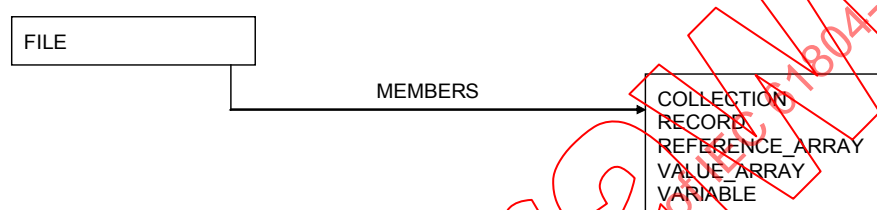


Figure 9 – FILE

IEC 1518/06

- **GRAPH** (graphique) décrit un graphique utilisé pour afficher des données provenant d'un dispositif dans l'application EDD (voir 7.16 et Figure 10).

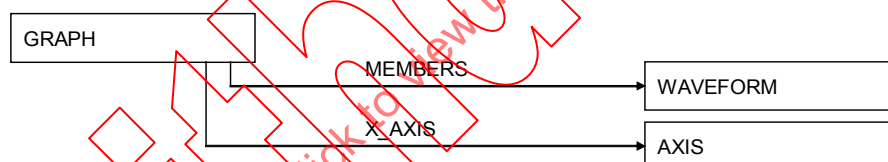
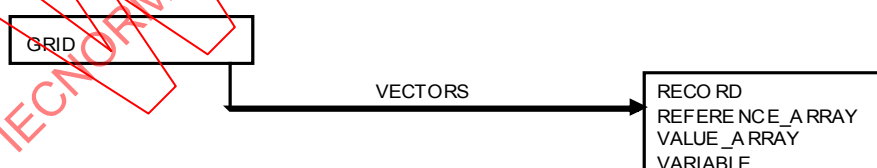


Figure 10 – GRAPH

IEC 1519/06

- **GRID** (grille) décrit une matrice utilisée pour afficher des données provenant d'un dispositif dans l'application EDD (voir 7.17 et Figure 11).



IEC 1520/06

Figure 11 – GRID

- **IMAGE** décrit une image visuelle (voir 7.18 et Figure 12).



Figure 12 – IMAGE

IEC 1521/06

- **IMPORT** (importation) est utilisé pour importer une EDD et modifier ses définitions existantes (voir 7.19).

- **LIKE** (similaire) crée une nouvelle instance d'une instance existante d'une construction de base. Les attributs de la nouvelle instance peuvent être redéfinis, ajoutés ou supprimés (voir 7.20 et Figure 13).

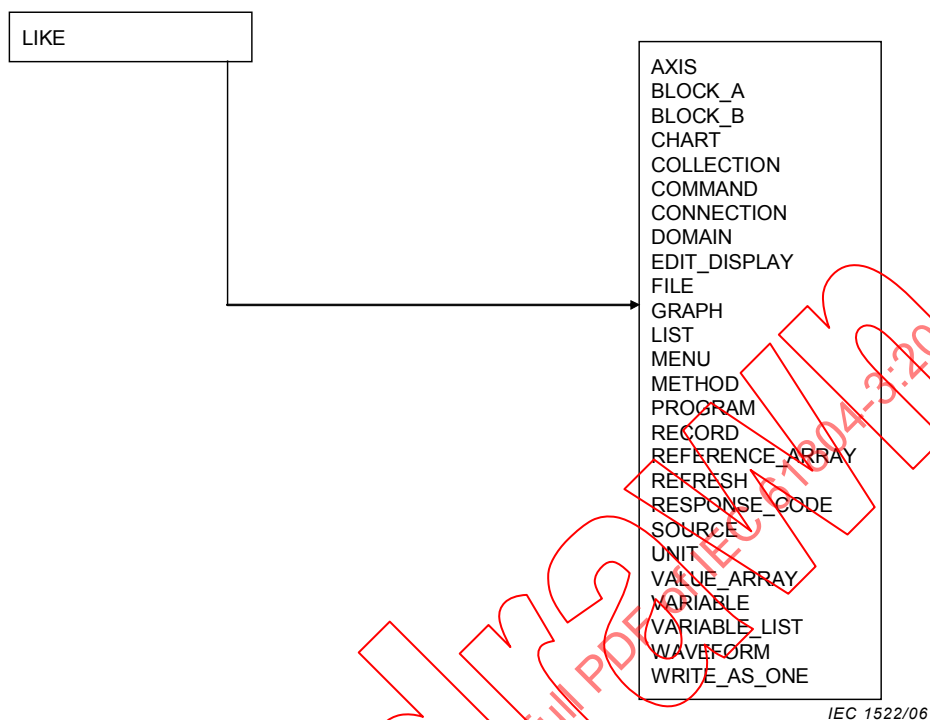


Figure 13 – LIKE

- **LIST** (liste) décrit une séquence d'éléments (voir 7.22 et Figure 14).

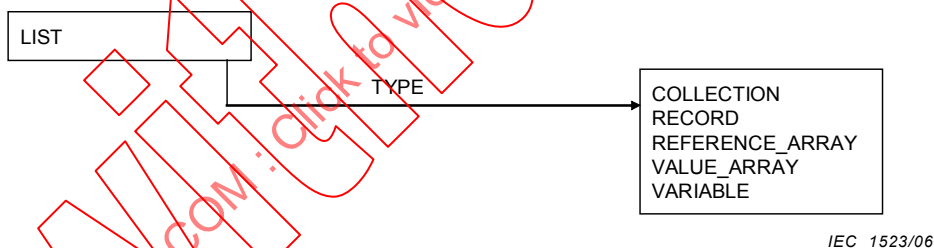


Figure 14 – LIST

- **MENU** décrit comment les données seront présentées à un utilisateur par une application EDD (voir 7.23 et Figure 15).

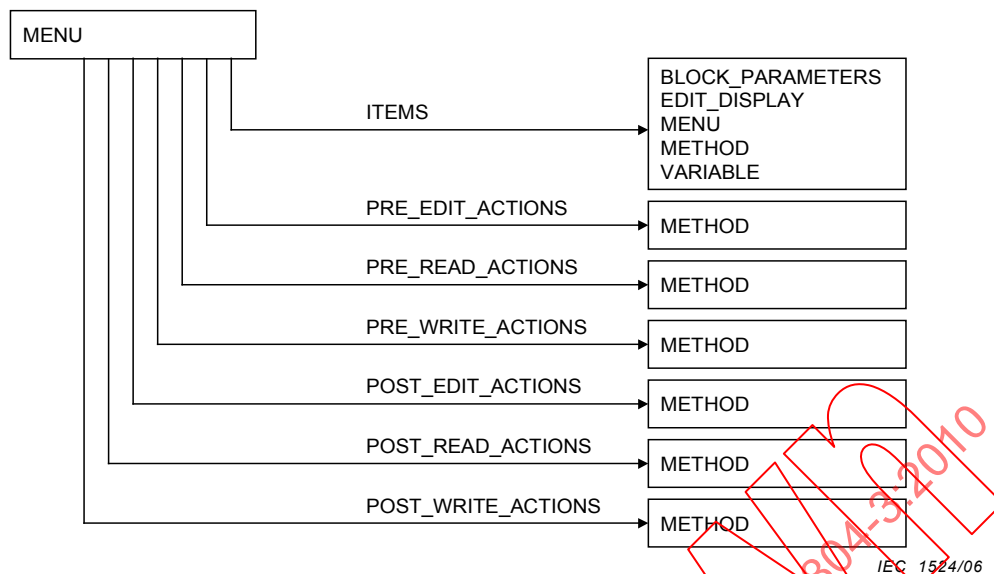


Figure 15 – MENU

- **METHOD** décrit l'exécution d'une séquence complexe d'interactions d'événements qui doit se produire entre les dispositifs du système tels que des unités d'affichage, des configurateurs et des dispositifs de terrain (voir 7.24).
- **PROGRAM** (programme) spécifie comment un code exécutable de dispositif peut être initié par un agent approprié (voir 7.25 et Figure 16).



Figure 16 – PROGRAM

- **RECORD** (enregistrement) décrit les données contenues dans le dispositif (voir 7.26 et Figure 17).

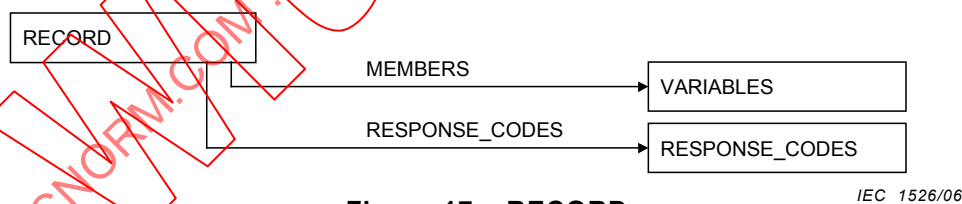
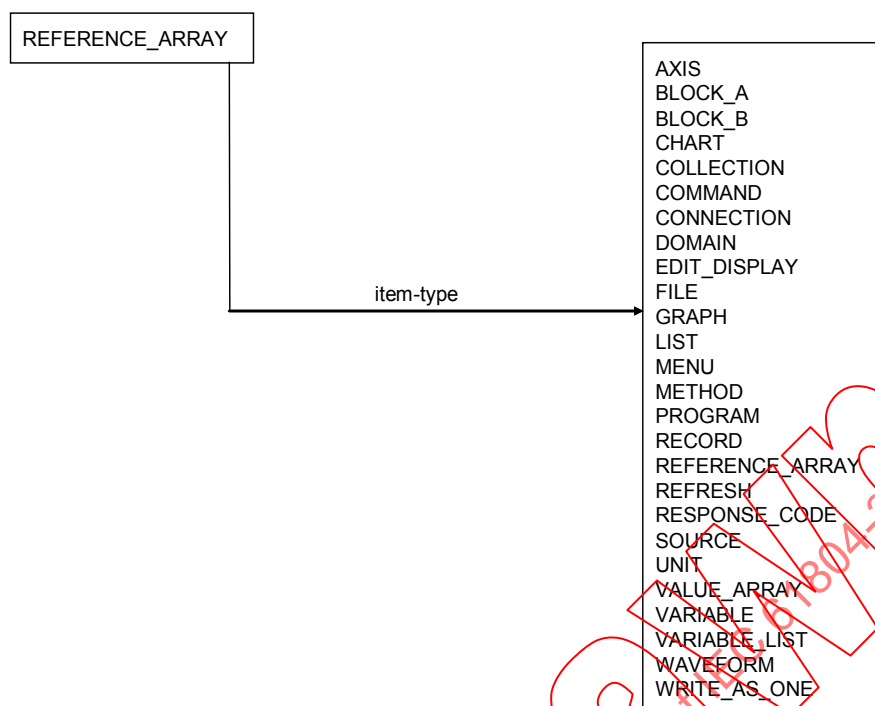


Figure 17 – RECORD

- **REFERENCE_ARRAY** (matrice de référence) est un ensemble d'éléments EDD du même type d'élément EDDL (par exemple, VARIABLE ou MENU). Un élément peut être référencé dans l'EDD via le nom de REFERENCE_ARRAY et l'index associé à l'élément (7.27 et Figure 18).



IEC 1527/06

Figure 18 – REFERENCE_ARRAY

- **Relations (voir 7.28):** trois types de relations sont spécifiés.

— REFRESH (réactualiser) décrit les relations entre les VARIABLE, les RECORD et les VALUE_ARRAY (voir 7.28.1 et Figure 19).



IEC 1528/06

Figure 19 – REFRESH

— UNIT (unité) décrit les relations entre les VARIABLE, les RECORD et les VALUE_ARRAY (voir 7.28.2 et Figure 20).



IEC 1529/06

Figure 20 – UNIT

— La relation WRITE_AS_ONE décrit les relations entre les VARIABLE, les RECORD et les VALUE_ARRAY (voir 7.28.3 et Figure 21).



IEC 1530/06

Figure 21 – WRITE_AS_ONE

- **RESPONSE_CODES** (codes de réponse) spécifie les codes de réponse spécifiques à l'application pour utilisation lors de la gestion d'un VARIABLE, RECORD, VALUE_ARRAY, VARIABLE_LIST, PROGRAM, ou DOMAIN (voir 7.29).
- **SOURCE** (voir 7.30 et Figure 22).

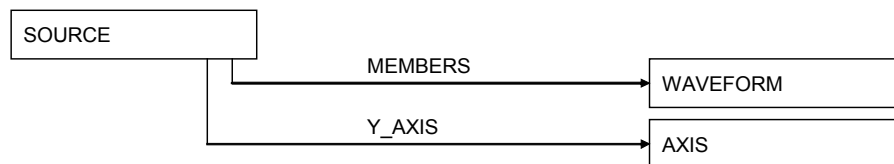


Figure 22 – SOURCE

IEC 1531/06

- **VALUE_ARRAY** (matrice de valeurs) décrit des groupes logiques de variables EDDL (voir 7.30.2.4 et Figure 23).

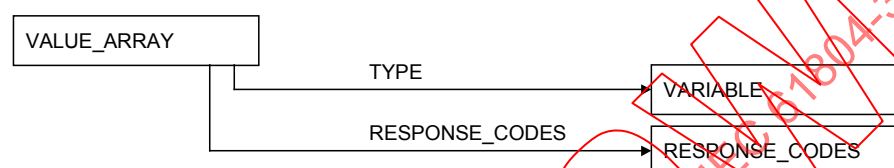


Figure 23 – VALUE_ARRAY

IEC 1532/06

- **VARIABLE** décrit les données contenues dans le dispositif (voir 7.33 et Figure 24).

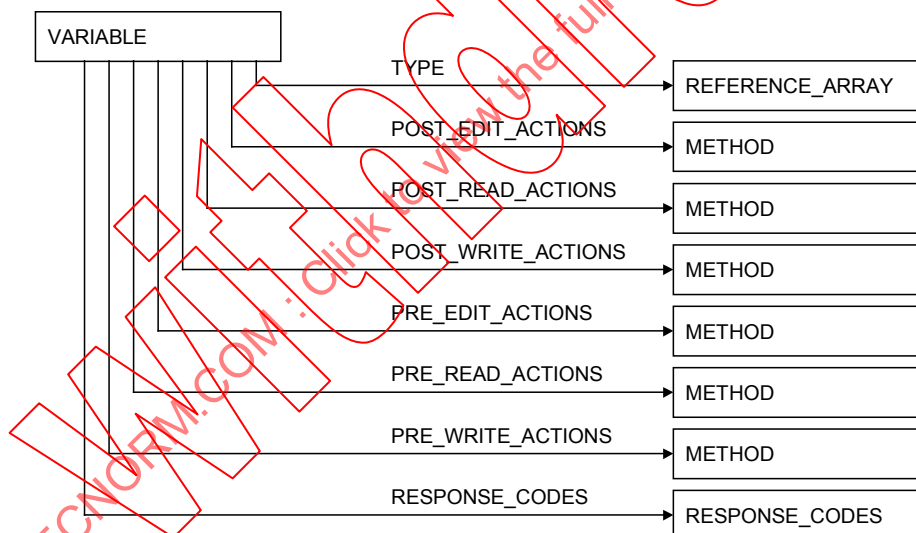


Figure 24 – VARIABLE

IEC 1533/06

- **VARIABLE_LIST** (liste de variables) décrit les regroupements logiques de données contenues dans le dispositif qui peuvent être communiqués sous forme de liste (voir 7.34 et Figure 25).

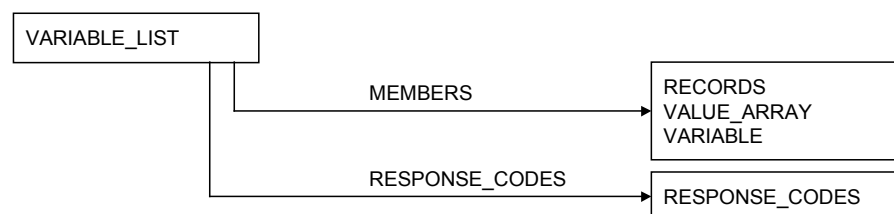
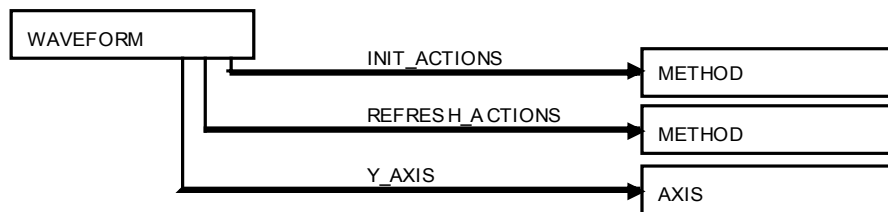


Figure 25 – VARIABLE_LIST

IEC 1534/06

- **WAVEFORM** (forme d'onde) (voir 7.35 et Figure 26).



IEC 1535/06

Figure 26 – WAVEFORM

7.1.5 Attributs communs

Les attributs communs suivants sont utilisés dans de nombreuses constructions de base:

- CLASSIFICATION
- COMPONENT_PARENT
- COMPONENT_PATH
- DEFINITION
- EMPHASIS
- HANDLING
- HEIGHT
- HELP
- LABEL
- LINE_COLOR
- LINE_TYPE
- MEMBERS
- PROTOCOL
- RESPONSE_CODES
- SUPPLIED_INTERFACE
- VALIDITY
- WIDTH

7.1.6 Éléments spéciaux

Les éléments spéciaux sont des mécanismes EDDL additionnels utilisés pour prendre en charge la gestion des fichiers, les instances multiples et la modification d'éléments de base EDDL, tels que:

- Les **expressions conditionnelles** qui spécifient des valeurs d'attributs, qui sont dépendantes des valeurs de variables à l'exécution
- Une **référence** qui est utilisée dans l'ensemble d'une description de dispositif pour désigner d'autres éléments dans le même EDD
- Une **expression** qui est une opération logique ou mathématique sur des valeurs de variable afin de modifier à l'exécution des valeurs d'attribut d'une construction de base

7.1.7 Règles pour les instances

Chaque instance d'une construction de base doit être identifiée par un identifiant de type chaîne de caractères. Cet identifiant doit être unique dans l'ensemble de l'EDD. Toutes les constructions de l'EDDL représentent des instances réelles d'informations du dispositif. Si un dispositif contient deux éléments d'information ou plus qui ont la même structure, la construction appropriée doit être répétée dans l'EDD avec des identifiants différents. En variante, la construction LIKE peut être utilisée pour créer une copie de la construction avec

un identifiant différent. L'identifiant dans la partie de définition lexicale n'est pas décrit à nouveau dans chaque construction.

7.1.8 Règles pour une liste de VARIABLES

Une liste de VARIABLE peut contenir des VARIABLE, des éléments de types composés ou des types composés. Les types composés sont:

- BLOCK_A PARAMETERS (paramètres de bloc A)
- COLLECTIONS (collections)
- RECORDS (enregistrements)
- REFERENCE_ARRAY (matrice de référence)
- VALUE_ARRAY (matrice de valeurs)
- VARIABLE_LISTS (listes de variables)

Au lieu de référencer chaque membre individuel d'un type composé, il est possible de faire référence à l'instance de type.

7.2 Informations d'identification d'une EDD

7.2.1 Structure générale

Objet

Les informations d'identification d'une EDD identifient de façon unique la description du dispositif d'un type de dispositif spécifique d'un fabricant de dispositif. Elles spécifient également la version de l'EDDL et le profil EDDL utilisé.

Ces informations d'identification doivent constituer la première entrée dans chaque EDD et doivent apparaître une seule fois.

Structure lexicale

DD_REVISION, DEVICE_REVISION, DEVICE_TYPE, EDD_PROFILE, EDD_VERSION,
MANUFACTURER, MANUFACTURER_EXT

NOTE 1 Les informations d'identification d'une EDD ne sont pas structurées de la même façon que d'autres constructions EDDL.

NOTE 2 L'ordre strict des mots clés des informations d'identification d'une EDD n'est pas impératif.

7.2.2 Attributs spécifiques

7.2.2.1 DD_REVISION

Objet

L'attribut DD_REVISION (révision DD) spécifie un code unique pour la révision d'une EDD du dispositif, il est défini par le fabricant.

NOTE 1 Il convient que le fabricant modifie le numéro de révision chaque fois qu'une nouvelle description de dispositif est émise pour un dispositif.

NOTE 2 Il peut exister plusieurs EDD pour une révision particulière d'un dispositif particulier, auquel cas l'attribut DD_REVISION permet de les distinguer.

Structure lexicale

DD_REVISION integer

L'attribut de DD_REVISION est spécifié dans le Tableau 2.

Tableau 2 – Attribut de DD_REVISION

Usage	Attribut	Description
m	integer	Entier non signé à deux octets contenant le code pour une révision de fichier EDD spécifique.

7.2.2.2 DEVICE_REVISION**Objet**

L'attribut DEVICE_REVISION (révision du dispositif) spécifie un code unique pour la révision de dispositif du dispositif de terrain, il est défini par le fabricant.

Structure lexicale

DEVICE_REVISION integer

L'attribut de DEVICE_REVISION est spécifié dans le Tableau 3.

Tableau 3 – Attribut de DEVICE_REVISION

Usage	Attribut	Description
m	integer	Entier non signé à deux octets contenant le code pour une révision du dispositif spécifique.

7.2.2.3 DEVICE_TYPE**Objet**

L'attribut DEVICE_TYPE spécifie un identifiant pour le type de dispositif, il est défini par le fabricant du dispositif. Il convient que l'identifiant soit unique pour chaque type.

NOTE Le type de dispositif peut spécifier une catégorie de dispositifs ou un produit spécifique.

Structure lexicale

DEVICE_TYPE (integer, identifiant)

Les attributs de DEVICE_TYPE sont spécifiés dans le Tableau 4.

Tableau 4 – Attributs de DEVICE_TYPE

Usage	Attribut	Description
s	integer	Entier non signé à deux octets contenant le code pour un type de dispositif ou groupe de types de dispositif spécifique.
s	identifiant	Référence à un entier à deux octets qui contient le code pour un type de dispositif ou groupe de types de dispositif spécifique.
NOTE Le mapping de l'identifiant à une constante entière peut être stocké dans un fichier séparé appelé "devices.cfg". Ce fichier contient une liste d'identifiants pour des dispositifs ou groupes de types de dispositif.		

7.2.2.4 EDD_PROFILE**Objet**

L'attribut EDD_PROFILE (profil EDD) spécifie le profil sélectionné de l'EDDL, il est défini dans l'Annexe D.

Structure lexicale

EDD_PROFILE integer

L'attribut de EDD_PROFILE est spécifié dans le Tableau 5.

Tableau 5 – Attribut de EDD_PROFILE

Usage	Attribut	Description
o	integer	Entier non signé à deux octets contenant le code pour les profils spécifiques sélectionnés dans l'Annexe D. Les codes suivants sont actuellement définis: 0x01 Profil pour HART Communication Foundation (voir D.4) 0x02 Profil pour Fieldbus Foundation (voir D.3) 0x03 Profil pour PROFIBUS (voir D.2) 0x04 Profil pour PROFINET (voir D.2)

7.2.2.5 EDD_VERSION**Objet**

L'attribut EDD_VERSION spécifie un code unique pour la version de l'EDD, qui a été utilisé par le fabricant dans la construction de l'EDD. Cet attribut doit être inclus avec toutes les descriptions de dispositif conformes à la présente norme.

Structure lexicale

EDD_VERSION integer

L'attribut de EDD_VERSION est spécifié dans le Tableau 6.

Tableau 6 – Attribut de EDD_VERSION

Usage	Attribut	Description
o	integer	Entier non signé à deux octets contenant le chiffre 1 pour la première édition de la norme.

7.2.2.6 MANUFACTURER**Objet**

L'attribut MANUFACTURER (fabricant) identifie le fabricant. Il convient que l'allocation de code soit gérée par les organisations spécifiques qui sont responsables des différents profils EDDL. La combinaison d'un EDD_PROFILE et d'un MANUFACTURER doit être unique.

Structure lexicale

MANUFACTURER (long integer, identifier)

Les attributs de MANUFACTURER sont spécifiés dans le Tableau 7.

Tableau 7 – Attributs de MANUFACTURER

Usage	Attribut	Description
s	long integer	Entier long non signé contenant le code pour un fabricant spécifique. Si tous les bits dans la valeur de l'entier sont définis à 1, la structure lexicale "MANUFACTURER_EXT" doit être présente et identifie le fabricant.
sm	identifier	Référence à un entier long non signé qui contient le code pour un fabricant spécifique.
NOTE Il convient que le mapping de l'identifiant avec une constante entière soit stocké dans un fichier séparé appelé "devices.cfg". Ce fichier contient une liste d'identifiants pour les fabricants.		

7.2.2.7 MANUFACTURER_EXT

Objet

L'attribut MANUFACTURER_EXT est réservé pour une utilisation future. C'est un espace réservé pour une chaîne permettant l'identification globale du MANUFACTURER (fabricant).

NOTE La spécification de cet attribut sera incluse lorsqu'elle sera établie au niveau international.

Structure lexicale

MANUFACTURER_EXT string

L'attribut de MANUFACTURER_EXT est spécifié dans le Tableau 8.

Tableau 8 – Attribut de MANUFACTURER_EXT

Usage	Attribut	Description
c	string	Chaîne contenant les informations pour l'identification mondiale unique d'un fabricant.

7.3 AXIS

7.3.1 Structure générale

Objet

AXIS (axe) décrit un axe d'un objet CHART (diagramme) ou GRAPH (graphique).

Structure lexicale

AXIS *identifieur* [CONSTANT_UNIT, HELP, LABEL, MAX_VALUE, MIN_VALUE, SCALING]

Les attributs d'AXIS sont spécifiés dans le Tableau 9.

Tableau 9 – Attributs d'AXIS

Usage	Attribut	Description
o	CONSTANT_UNIT	Elément lexical (voir 7.3.2.1).
o	HELP	Elément lexical (voir 7.36.8).
o	LABEL	Elément lexical (voir 7.36.9).
o	MAX_VALUE	Elément lexical (voir 7.3.2.1).
o	MIN_VALUE	Elément lexical (voir 7.3.2.3).
o	SCALING	Elément lexical (voir 7.3.2.4).

7.3.2 Attributs spécifiques

7.3.2.1 CONSTANT_UNIT

Objet

L'attribut CONSTANT_UNIT (unité constante) est utilisé, si un AXIS a un code unitaire qui ne change jamais. L'attribut CONSTANT_UNIT est spécifié sous la forme d'une chaîne qui est affichée avec la valeur. Un AXIS (axe) sans unité constante n'a pas d'unités associées ou bien les unités ne sont pas constantes.

Structure lexicale

CONSTANT_UNIT (string)<exp>

L'attribut de CONSTANT_UNIT est spécifié dans le Tableau 175.

7.3.2.2 MAX_VALUE

Objet

L'attribut MAX_VALUE spécifie la valeur associée à la dernière marque de graduation de l'AXIS (axe). La dernière marque de graduation d'un X_AXIS et un Y_AXIS est la marque de graduation la plus à droite et la plus haute, respectivement. Par défaut, la valeur associée à la dernière marque de graduation d'un AXIS sans attribut MAX_VALUE est déduite des WAVEFORM (forme d'onde) sur le GRAPH (graphique).

Structure lexicale

MAX_VALUE (reference, double, float, integer, unsigned_integer)<exp>

Les attributs de MAX_VALUE sont spécifiés dans le Tableau 10.

Tableau 10 – Attribut de MAX_VALUE, MIN_VALUE

Usage	Attribut	Description
o	reference	Référence à une instance de VARIABLE, un membre d'une instance de RECORD, un élément d'une instance de VALUE_ARRAY ou un élément d'une instance de REFERENCE_ARRAY.
o	double	Constante à virgule flottante à double précision.
o	float	Constante à virgule flottante à simple précision.
o	integer	Constante entière.
o	unsigned_integer	Constante entière non signée.

7.3.2.3 MIN_VALUE

Objet

L'attribut MIN_VALUE (valeur minimale) spécifie la valeur associée à la première marque de graduation de l'AXIS (axe). La première marque de graduation d'un X_AXIS et un Y_AXIS est la marque de graduation la plus à gauche et la plus basse, respectivement. Par défaut, la valeur associée à la première marque de graduation d'un AXIS (axe) sans attribut MIN_VALUE est déduite des WAVEFORM (forme d'onde) sur le GRAPH (graphique).

Structure lexicale

MIN_VALUE (reference, double, float, integer, unsigned_integer)<exp>

Les attributs de MIN_VALUE sont spécifiés dans le Tableau 10.

7.3.2.4 SCALING

Objet

L'attribut SCALING (mise à l'échelle) spécifie la façon selon laquelle il convient que les marques de graduation de l'AXIS soient mises à l'échelle. Par défaut, un AXIS (axe) sans attribut SCALING est mis à l'échelle de façon linéaire.

Structure lexicale

SCALING (LINEAR, LOGARITHMIC)<exp>

Les attributs de SCALING sont spécifiés dans le Tableau 11.

Tableau 11 – Attribut de SCALING

Usage	Attribut	Description
s	LINEAR	Il convient que les marques de graduation sur l'axe soient mises à l'échelle de façon linéaire.
s	LOGARITHMIC	Il convient que les marques de graduation sur l'axe soient mises à l'échelle de façon logarithmique (base 10).

7.4 BLOCK

7.4.1 BLOCK_A

7.4.1.1 Structure générale

Objet

Un dispositif peut être organisé en blocs logiques. Chaque BLOCK_A est un regroupement logique de PARAMETERS (paramètres) et d'informations additionnelles pour des applications BLOCK_A. Les types de blocs sont décrits par l'attribut CHARACTERISTICS. Plusieurs BLOCK_A du même type peuvent être instanciés.

NOTE Cette approche du BLOCK_A est optimisée pour l'efficacité d'accès.

Structure lexicale

BLOCK_A identifieur [CHARACTERISTICS, LABEL, PARAMETERS, AXIS_ITEMS, CHART_ITEMS, COLLECTION_ITEMS, EDIT_DISPLAY_ITEMS, FILE_ITEMS, GRAPH_ITEMS, GRID_ITEMS, HELP, IMAGE_ITEMS, LIST_ITEMS, MENU_ITEMS, METHOD_ITEMS, PARAMETER_LISTS, REFERENCE_ARRAY_ITEMS, REFRESH_ITEMS, SOURCE_ITEMS, UNIT_ITEMS, WAVEFORM_ITEMS, WRITE_AS_ONE_ITEMS, CHARTS, LISTS, GRAPHS, GRIDS, MENUS, METHODS]

Les attributs de BLOCK_A sont spécifiés dans le Tableau 12.

Tableau 12 – Attributs de BLOCK_A

Usage	Attribut	Description
m	CHARACTERISTICS	Elément lexical (voir 7.4.1.2.1).
m	LABEL	Elément lexical (voir 7.36.9).
m	PARAMETERS	Elément lexical (voir 7.4.1.2.2).
o	AXIS_ITEMS	Elément lexical (voir 7.4.1.2.3).
o	CHART_ITEMS	Elément lexical (voir 7.4.1.2.4).
o	COLLECTION_ITEMS	Elément lexical (voir 7.4.1.2.5).
o	EDIT_DISPLAY_ITEMS	Elément lexical (voir 7.4.1.2.6).
o	FILE_ITEMS	Elément lexical (voir 7.4.1.2.7).
o	GRAPH_ITEMS	Elément lexical (voir 7.4.1.2.8).
o	GRID_ITEMS	Éléments lexicaux (voir 7.4.1.2.9).
o	HELP	Elément lexical (voir 7.36.8).
o	IMAGE_ITEMS	Éléments lexicaux (voir 7.4.1.2.10).
o	LIST_ITEMS	Elément lexical (voir 7.4.1.2.11).
o	LOCAL_PARAMETERS	Elément lexical (voir 7.4.1.2.12).
o	MENU_ITEMS	Elément lexical (voir 7.4.1.2.13).
o	METHOD_ITEMS	Elément lexical (voir 7.4.1.2.14).
o	PARAMETER_LISTS	Elément lexical (voir 7.4.1.2.15).
o	REFERENCE_ARRAY_ITEMS	Elément lexical (voir 7.4.1.2.16).
o	REFRESH_ITEMS	Elément lexical (voir 7.4.1.2.17).
o	SOURCE_ITEMS	Elément lexical (voir 7.4.1.2.18).
o	UNIT_ITEMS	Elément lexical (voir 7.4.1.2.19).
o	WAVEFORM_ITEMS	Elément lexical (voir 7.4.1.2.20).
o	WRITE_AS_ONE_ITEMS	Elément lexical (voir 7.4.1.2.21).
o	CHARTS	Elément lexical (voir 7.4.1.2.22).
o	LISTS	Elément lexical (voir 7.4.1.2.23).
o	GRAPHS	Elément lexical (voir 7.4.1.2.24).
o	GRIDS	Elément lexical (voir 7.4.1.2.25).
o	MENUS	Elément lexical (voir 7.4.1.2.26).
o	METHODS	Elément lexical (voir 7.4.1.2.27).

7.4.1.2 Attributs spécifiques

7.4.1.2.1 CHARACTERISTICS

Objet

L'attribut CHARACTERISTICS (caractéristiques) spécifie les caractéristiques externes pour un BLOCK_A. CHARACTERISTICS peut comprendre une classe, un type de fonction, un type de bloc et un temps d'exécution. Les caractéristiques externes d'un BLOCK_A sont contenues dans un RECORD (enregistrement).

NOTE En général, les CHARACTERISTICS (caractéristiques) de BLOCK_A fixent le type du BLOCK_A.

Structure lexicale

CHARACTERISTICS *reference*

L'attribut de CHARACTERISTICS est spécifié dans le Tableau 13.

Tableau 13 – Attribut de CHARACTERISTICS

Usage	Attribut	Description
m	reference	Référence à une instance de RECORD (enregistrement).

7.4.1.2.2 PARAMETERS**Objet**

L'attribut PARAMETERS (paramètres) contient une liste de références à des instances de VARIABLE, VALUE_ARRAY (matrice de valeurs) ou RECORD (enregistrement). Chaque référence doit avoir un identifiant et peut avoir une description et/ou un texte d'aide.

NOTE Les PARAMETERS dans un BLOCK_A correspondent à des objets de communication individuels, répertoriés dans un dictionnaire d'objets du dispositif. Les éléments PARAMETERS font référence aux définitions de communication de ces objets. Le dictionnaire d'objets localise les emplacements spécifiques des données dans un dispositif.

Structure lexicale

PARAMETERS [identifier, reference, description, help] +

Les attributs de PARAMETERS sont spécifiés dans le Tableau 14.

Tableau 14 – Attributs de PARAMETERS

Usage	Attribut	Description
m	identifier	Identifiant de VARIABLE, VALUE_ARRAY ou RECORD. Chaque élément de la liste PARAMETERS doit avoir un identifiant utilisé dans l'EDD pour faire référence à celle-ci.
m	reference	Référence à une instance de VARIABLE, VALUE_ARRAY ou RECORD.
o	description	Breve description pour la référence.
o	help	Texte d'aide pour la référence.

7.4.1.2.3 AXIS_ITEMS**Objet**

L'attribut AXIS_ITEMS (éléments d'axe) est une liste des instances d'AXIS (axe) associées à un BLOCK_A.

Structure lexicale

AXIS_ITEMS reference +

L'attribut d'AXIS_ITEMS est spécifié dans le Tableau 15.

Tableau 15 – Attribut d'AXIS_ITEMS

Usage	Attribut	Description
m	reference	Référence à une instance d'AXIS.

7.4.1.2.4 CHART_ITEMS**Objet**

L'attribut CHART_ITEMS (éléments de diagramme) est une liste des instances de CHART (diagramme) associées à un BLOCK_A.

Structure lexicale

CHART_ITEMS reference+

L'attribut de CHART_ITEMS est spécifié dans le Tableau 16.

Tableau 16 – Attribut de CHART_ITEMS

Usage	Attribut	Description
m	reference	Référence à une instance de CHART.

7.4.1.2.5 COLLECTION_ITEMS

Objet

L'attribut COLLECTION_ITEMS (éléments de collection) est une liste des instances de COLLECTION associées à un BLOCK_A.

Structure lexicale

COLLECTION_ITEMS reference+

L'attribut de COLLECTION_ITEMS est spécifié dans le Tableau 17.

Tableau 17 – Attribut de COLLECTION_ITEMS

Usage	Attribut	Description
m	reference	Référence à une instance de COLLECTION.

7.4.1.2.6 EDIT_DISPLAY_ITEMS

Objet

L'attribut EDIT_DISPLAY_ITEMS est une liste des instances d'EDIT_DISPLAY associées à BLOCK_A.

Structure lexicale

EDIT_DISPLAY_ITEMS reference+

L'attribut d'EDIT_DISPLAY_ITEMS est spécifié dans le Tableau 18.

Tableau 18 – Attribut d'EDIT_DISPLAY_ITEMS

Usage	Attribut	Description
m	reference	Référence à une instance d'EDIT_DISPLAY.

7.4.1.2.7 FILE_ITEMS

Objet

L'attribut FILE_ITEMS (éléments de fichier) est une liste des instances de FILE (fichier) associées à un BLOCK_A.

Structure lexicale

FILE_ITEMS reference+

L'attribut de FILE_ITEMS est spécifié dans le Tableau 19.

Tableau 19 – Attribut de FILE_ITEMS

Usage	Attribut	Description
m	reference	Référence à une instance de FILE.

7.4.1.2.8 GRAPH_ITEMS**Objet**

L'attribut GRAPH_ITEMS (éléments de graphique) est une liste des instances de GRAPH (graphique) associées à un BLOCK_A.

Structure lexicale

GRAPH_ITEMS reference+

L'attribut de GRAPH_ITEMS est spécifié dans le Tableau 20.

Tableau 20 – Attribut de GRAPH_ITEMS

Usage	Attribut	Description
m	reference	Référence à une instance de GRAPH.

7.4.1.2.9 GRID_ITEMS**Objet**

L'attribut GRID_ITEMS (éléments de grille) est une liste d'instances de GRID (grille) associées à un BLOCK_A.

Structure lexicale

GRID_ITEMS reference+

L'attribut de GRID_ITEMS est spécifié dans le Tableau 21.

Tableau 21 – Attribut de GRID_ITEMS

Usage	Attribut	Description
m	reference	Référence à une instance de GRID.

7.4.1.2.10 IMAGE_ITEMS**Objet**

L'attribut IMAGE_ITEMS (éléments d'image) est une liste des instances d'IMAGE associées à BLOCK_A.

Structure lexicale

IMAGE_ITEMS reference+

L'attribut d'IMAGE_ITEMS est spécifié dans le Tableau 22.

Tableau 22 – Attribut d'IMAGE_ITEMS

Usage	Attribut	Description
m	reference	Référence à une instance d'IMAGE.

7.4.1.2.11 LIST_ITEMS

Objet

L'attribut LIST_ITEMS (éléments de liste) est une liste des instances de LIST (liste) associées à BLOCK_A.

Structure lexicale

LIST_ITEMS reference+

L'attribut de LIST_ITEMS est spécifié dans le Tableau 23.

Tableau 23 – Attribut de LIST_ITEMS

Usage	Attribut	Description
m	reference	Référence à une instance de LIST.

7.4.1.2.12 LOCAL_PARAMETERS

Objet

L'attribut LOCAL_PARAMETERS (paramètres locaux) contient une liste de références à des instances de VARIABLE, VALUE_ARRAY ou RECORD. Chaque référence doit avoir un identifiant et peut avoir une description et/ou un texte d'aide.

NOTE Les LOCAL_PARAMETERS dans un BLOCK_A ne correspondent pas à des objets de communication; ce sont des paramètres qui sont utilisés uniquement par l'application EDD.

Structure lexicale

LOCAL_PARAMETERS [identifier, reference, description, help] +

Les attributs de LOCAL_PARAMETERS sont spécifiés dans le Tableau 14.

7.4.1.2.13 MENU_ITEMS

Objet

L'attribut MENU_ITEMS spécifie les éléments MENU associés à BLOCK_A.

Structure lexicale

MENU_ITEMS reference+

L'attribut de MENU_ITEMS est spécifié dans le Tableau 24.

Tableau 24 – Attribut de MENU_ITEMS

Usage	Attribut	Description
m	reference	Référence à une instance de MENU.

7.4.1.2.14 METHOD_ITEMS

Objet

L'attribut METHOD_ITEMS (éléments de méthode) spécifie les METHOD associés à un BLOCK_A.

Structure lexicale

METHOD_ITEMS reference+

L'attribut de METHOD_ITEMS est spécifié dans le Tableau 25.

Tableau 25 – Attribut de METHOD_ITEMS

Usage	Attribut	Description
m	reference	Référence à une instance de METHOD.

7.4.1.2.15 PARAMETER_LISTS

Objet

L'attribut PARAMETER_LISTS (listes de paramètres) spécifie les VARIABLE_LIST (liste de variables) associés à un BLOCK_A, ainsi qu'une description optionnelle et une aide pour chaque VARIABLE_LIST. Chaque BLOCK_A peut contenir plusieurs PARAMETER_LISTS.

Structure lexicale

PARAMETER_LISTS [identifier, reference, description, help] +

Les attributs de PARAMETER_LISTS sont spécifiés dans le Tableau 26.

Tableau 26 – Attributs de PARAMETER_LISTS

Usage	Attribut	Description
m	identifier	Identifiant de la VARIABLE_LIST. Chaque élément de la liste PARAMETER_LIST doit avoir un identifiant devant être utilisé dans l'EDD pour faire référence à celle-ci.
m	reference	Référence à une instance de VARIABLE_LIST.
o	description	Brève description de l'élément.
o	help	Texte d'aide pour l'élément.

7.4.1.2.16 REFERENCE_ARRAY_ITEMS

Objet

L'attribut REFERENCE_ARRAY_ITEMS (éléments de matrice de référence) spécifie les REFERENCE_ARRAY (matrice de référence) associés à un BLOCK_A.

Structure lexicale

REFERENCE_ARRAY_ITEMS reference +

L'attribut de REFERENCE_ARRAY_ITEMS est spécifié dans le Tableau 27.

Tableau 27 – Attribut de REFERENCE_ARRAY_ITEMS

Usage	Attribut	Description
m	reference	Référence à une instance de REFERENCE_ARRAY.

7.4.1.2.17 REFRESH_ITEMS

Objet

L'attribut REFRESH_ITEMS (éléments de REFRESH) spécifie les relations REFRESH (réactualiser) associées à un BLOCK_A.

Structure lexicale

REFRESH_ITEMS reference +

L'attribut de REFRESH_ITEMS est spécifié dans le Tableau 28.

Tableau 28 – Attribut de REFRESH_ITEMS

Usage	Attribut	Description
m	reference	Référence à une instance de REFRESH.

7.4.1.2.18 SOURCE_ITEMS

Objet

L'attribut SOURCE_ITEMS (éléments source) spécifie les instances de SOURCE associées à un BLOCK_A.

Structure lexicale

SOURCE_ITEMS reference+

L'attribut de SOURCE_ITEMS est spécifié dans le Tableau 29.

Tableau 29 – Attribut de SOURCE_ITEMS

Usage	Attribut	Description
m	reference	Référence à une instance de SOURCE.

7.4.1.2.19 UNIT_ITEMS

Objet

L'attribut UNIT_ITEMS (éléments d'unité) spécifie les relations des UNIT (unité) associés à un BLOCK_A.

Structure lexicale

UNIT_ITEMS reference+

L'attribut de UNIT_ITEMS est spécifié dans le Tableau 30.

Tableau 30 – Attribut de UNIT_ITEMS

Usage	Attribut	Description
m	reference	Référence à une instance d'UNIT.

7.4.1.2.20 WAVEFORM_ITEMS

Objet

L'attribut WAVEFORM_ITEMS (éléments de forme d'onde) spécifie les instances de WAVEFORM (forme d'onde) associées à un BLOCK_A.

Structure lexicale

WAVEFORM_ITEMS reference+

L'attribut de WAVEFORM_ITEMS est spécifié dans le Tableau 31.

Tableau 31 – Attribut de WAVEFORM_ITEMS

Usage	Attribut	Description
m	reference	Référence à une instance de WAVEFORM.

7.4.1.2.21 WRITE_AS_ONE_ITEMS

Objet

L'attribut WRITE_AS_ONE_ITEMS (éléments de WRITE_AS_ONE) spécifie les relations WRITE_AS_ONE associées à un BLOCK_A.

Structure lexicale

WRITE_AS_ONE_ITEMS reference+

L'attribut de WRITE_AS_ONE_ITEMS est spécifié dans le Tableau 32.

Tableau 32 – Attribut de WRITE_AS_ONE_ITEMS

Usage	Attribut	Description
m	reference	Référence à une instance de WRITE_AS_ONE.

7.4.1.2.22 CHARTS

Objet

L'attribut CHARTS (diagrammes) spécifie les éléments CHART (diagramme) associés à un BLOCK_A. Seuls les éléments CHART qui sont référencés via une instance spécifique de BLOCK_A (voir 7.38.21) sont à spécifier.

Structure lexicale

CHARTS (identifiant, reference, description, help)+

L'attribut de CHARTS est spécifié dans le Tableau 33.

Tableau 33 – Attribut de CHARTS

Usage	Attribut	Description
m	identifiant	Nom par lequel le CHART peut être référencé.
m	reference	Référence à une instance de CHART.
o	description	Brève description du CHART.
o	help	Texte d'aide pour le CHART.

7.4.1.2.23 LISTS

Objet

L'attribut LISTS spécifie les objets LIST associés à un BLOCK_A. Seuls les LIST qui sont référencés via une instance spécifique de BLOCK_A (voir 7.4.1) sont à spécifier.

Structure lexicale

LISTS (identifiant, reference, description, help)+

Les attributs de LISTS sont spécifiés dans le Tableau 34.

Tableau 34 – Attributs de LISTS

Usage	Attribut	Description
m	identifiant	Nom par lequel l'objet LIST peut être référencé.
m	reference	Référence à une instance de LIST.

Usage	Attribut	Description
o	description	Brève description de l'objet LIST.
o	help	Texte d'aide pour l'objet LIST.

7.4.1.2.24 GRAPHS

Objet

L'attribut GRAPHS (graphiques) spécifie les objets GRAPH associés à un BLOCK_A. Seuls les GRAPH qui sont référencés via une instance spécifique de BLOCK_A (voir 7.38.23) sont à spécifier.

Structure lexicale

GRAPHS (identifier, reference, description, help)+

Les attributs de GRAPHS sont spécifiés dans le Tableau 35.

Tableau 35 – Attributs de GRAPHS

Usage	Attribut	Description
m	identifier	Nom par lequel l'objet GRAPH peut être référencé.
m	reference	Référence à une instance de GRAPH.
o	description	Brève description du GRAPH.
o	help	Texte d'aide pour le GRAPH.

7.4.1.2.25 GRIDS

Objet

L'attribut GRIDS (grilles) spécifie les objets GRID associés à un BLOCK_A. Seuls les GRID qui sont référencés via une instance spécifique de BLOCK_A (voir 7.38.24) sont à spécifier.

Structure lexicale

GRIDS (identifier, reference, description, help)+

Les attributs de GRIDS sont spécifiés dans le Tableau 36.

Tableau 36 – Attributs de GRIDS

Usage	Attribut	Description
m	identifier	Nom par lequel l'objet GRID peut être référencé.
m	reference	Référence à une instance de GRID.
o	description	Brève description du GRID.
o	help	Texte d'aide pour le GRID.

7.4.1.2.26 MENUS

Objet

L'attribut MENUS spécifie les objets MENU associés à un BLOCK_A. Seuls les MENU qui sont référencés via une instance spécifique de BLOCK_A (voir 7.38.25) sont à spécifier.

Structure lexicale

MENUS (identifier, reference, description, help)+

Les attributs de MENUS sont spécifiés dans le Tableau 37.

Tableau 37 – Attributs de MENUS

Usage	Attribut	Description
m	identifiant	Nom par lequel l'objet MENU peut être référencé.
m	reference	Référence à une instance de MENU.
o	description	Brève description du MENU.
o	help	Texte d'aide pour le MENU.

7.4.1.2.27 METHODS

Objet

L'attribut METHODS spécifie les éléments METHOD associés au BLOCK_A. Seuls les METHOD qui sont référencés via une instance spécifique de BLOCK_A (voir 7.38.26) sont à spécifier.

Structure lexicale

METHODS (identifiant, reference, description, help)*

Les attributs de METHODS sont spécifiés dans le Tableau 38.

Tableau 38 – Attributs de METHODS

Usage	Attribut	Description
m	identifiant	Nom par lequel l'objet METHOD peut être référencé.
m	reference	Référence à une instance de METHOD.
o	description	Brève description de l'objet METHOD.
o	help	Texte d'aide pour l'objet METHOD.

7.4.2 BLOCK_B

7.4.2.1 Structure générale

Objet

Les variables d'un dispositif ou module sont respectivement structurées en blocs correspondant aux composants ou aux parties fonctionnelles d'un dispositif. Trois types de blocs sont définis: PHYSICAL (physique), TRANSDUCER (transducteur) et FUNCTION (fonction). Plusieurs BLOCK_B du même TYPE peuvent être instanciés. Afin de faciliter l'accès, chaque instance a son propre NUMBER (numéro). Pour chacun des types de bloc, il existe différentes chaînes de séquences de NUMBER (numéro), commençant par 1.

Lors de l'accès à un VARIABLE dans un dispositif, la construction de BLOCK_B est utilisée conjointement avec COMMAND (commande) pour permettre un adressage relatif.

NOTE 1 Cette approche basée sur BLOCK_B est optimisée pour une meilleure efficacité de la mémoire.

NOTE 2 Le stockage d'une variable dans un dispositif est spécifique au fabricant et est représenté par un répertoire de dispositifs. Un répertoire de dispositifs contient un résumé des entrées de BLOCK_B disponibles dans un dispositif. De plus, le répertoire de dispositifs contient pour chaque BLOCK_B un pointeur vers son adresse physique. Une application EDD trouve un objet unique en ajoutant un décalage à l'adresse physique de BLOCK_B.

NOTE 3 Dans une instance de BLOCK_B, un indexage relatif est utilisé. Cet index relatif est défini dans le profil de dispositif.

Structure lexicale

BLOCK_B identifier [NUMBER, TYPE]

Les attributs de BLOCK_B sont spécifiés dans le Tableau 39.

Tableau 39 – Attributs de BLOCK_B

Usage	Attribut	Description
m	NUMBER	Élément lexical (voir 7.4.2.2.1).
m	TYPE	Élément lexical (voir 7.4.2.2.2).

7.4.2.2 Attributs spécifiques

7.4.2.2.1 NUMBER

Objet

L'attribut NUMBER (numéro) énumère chaque BLOCK_B du même TYPE dans le dispositif. Un dispositif peut contenir plusieurs instances de BLOCK_B, qui sont identifiées avec l'attribut NUMBER.

Structure lexicale

NUMBER (integer, expression)<exp>

L'attribut de NUMBER est spécifié dans le Tableau 40.

Tableau 40 – Attributs de NUMBER

Usage	Attribut	Description
s	integer	Numéro de l'instance de BLOCK_B du même TYPE.
s	expression	Expression qui doit être évaluée comme un entier non négatif qui est dans l'intervalle d'index de l'attribut NUMBER (pour la description des expressions, voir 7.39.7).

7.4.2.2.2 TYPE

Objet

L'attribut TYPE est utilisé pour spécifier un des trois types de bloc.

Structure lexicale

TYPE [PHYSICAL, TRANSDUCER, FUNCTION]

Les attributs de TYPE sont spécifiés dans le Tableau 41.

Tableau 41 – Attributs de TYPE

Usage	Attribut	Description
s	FUNCTION	Bloc nommé constitué d'un ou plusieurs paramètres d'entrée, de sortie ou internes. Les fonctions blocs représentent les fonctions d'automatisation de base exécutées par une application EDD, qui est indépendante des dispositifs spécifiques et du réseau. Chaque fonction bloc traite des paramètres d'entrée conformément à un algorithme spécifié et un ensemble de paramètres internes. Ils génèrent des paramètres de sortie qui sont disponibles pour utilisation dans la même application EDD de fonction bloc ou par d'autres applications de fonction bloc.
s	PHYSICAL	Les caractéristiques spécifiques du matériel/ressource d'un dispositif de terrain sont rendues visibles par l'intermédiaire du bloc physique. De manière similaire aux blocs transducteur, ils isolent des fonctions blocs du matériel physique en contenant un ensemble de paramètres du matériel/ressource indépendants de l'implémentation.
s	TRANSDUCER	Les blocs transducteur isolent les fonctions blocs des éléments spécifiques du dispositif E/S, tels que des capteurs, des actionneurs et des commutateurs. Les blocs transducteur contrôlent l'accès à des dispositifs E/S par l'intermédiaire d'une interface indépendante des dispositifs définie pour une utilisation par des fonctions blocs. Les blocs transducteur remplissent des fonctions, telles que l'étalonnage et la linéarisation, sur des données E/S pour convertir celles-ci en représentation indépendante du dispositif.

7.5 CHART

7.5.1 Structure générale

Objet

CHART décrit un diagramme utilisé pour afficher des données provenant d'un dispositif dans l'application EDD. Un objet CHART est utilisé pour afficher des valeurs de données continues provenant du dispositif, par opposition à un GRAPH (graphique) qui est utilisé pour afficher un ensemble de données qui est stocké dans un dispositif. Les données provenant du dispositif peuvent être ajustées avant d'être affichées via l'utilisation d'INIT_ACTIONS (actions d'initialisation) et REFRESH_ACTIONS (actions de réactualisation) sur les membres SOURCE du CHART.

Structure lexicale

CHART identifier [MEMBERS, CYCLE_TIME, HEIGHT, WIDTH, LENGTH, HELP, LABEL, TYPE, VALIDITY]

Les attributs de CHART sont spécifiés dans le Tableau 42.

Tableau 42 – Attributs de CHART

Usage	Attribut	Description
M	MEMBERS	Elément lexical (voir 7.36.12).
o	CYCLE_TIME	Elément lexical (voir 7.5.2.1).
o	HEIGHT	Elément lexical (voir 7.36.7).
o	HELP	Elément lexical (voir 7.36.8).
o	LABEL	Elément lexical (voir 7.36.9).
o	LENGTH	Elément lexical (voir 7.5.2.2).
o	TYPE	Elément lexical (voir 7.5.2.3).
o	VALIDITY	Elément lexical (voir 7.36.16).
o	WIDTH	Elément lexical (voir 7.36.15).

7.5.2 Attributs spécifiques

7.5.2.1 CYCLE_TIME

Objet

L'attribut CYCLE_TIME (durée de cycle) spécifie le temps, en ms, durant lequel l'application EDD doit attendre entre des lectures successives des valeurs sources du dispositif. Par défaut, la durée de cycle d'un CHART (diagramme) sans attribut CYCLE_TIME est de 1 000 ms.

Structure lexicale

CYCLE_TIME (integer) <exp>

L'attribut de CYCLE_TIME est spécifié dans le Tableau 43.

Tableau 43 – Attribut de CYCLE_TIME

Usage	Attribut	Description
m	integer	Temps, en ms, entre des lectures successives des valeurs sources du dispositif.

7.5.2.2 LENGTH

Objet

L'attribut LENGTH (durée) spécifie la durée, en ms, qui est affichée par le CHART. Le nombre d'échantillons affichés par un diagramme peut être calculé en divisant l'attribut LENGTH par l'attribut CYCLE_TIME (durée de cycle). Par défaut, la durée de cycle d'un CHART sans attribut LENGTH est de 600 000 ms.

Structure lexicale

LENGTH (integer) <exp>

L'attribut de LENGTH est spécifié dans le Tableau 44.

Tableau 44 – Attribut de LENGTH

Usage	Attribut	Description
m	integer	Entier représentant la durée, en ms, qui est affichée par le CHART.

7.5.2.3 TYPE

Objet

L'attribut TYPE spécifie le type de CHART (diagramme) que l'EDD doit afficher. Par défaut, le type d'un CHART sans attribut TYPE est STRIP.

Structure lexicale

TYPE (GAUGE, HORIZONTAL_BAR, SCOPE, STRIP, SWEEP, VERTICAL_BAR) <exp>

Les attributs de TYPE sont spécifiés dans le Tableau 45.

Tableau 45 – Attribut de TYPE

Usage	Attribut	Description
s	GAUGE	Une valeur source unique est affichée sous forme de jauge, qui est une représentation graphique des données similaire à la jauge de carburant d'une automobile.

Usage	Attribut	Description
s	HORIZONTAL_BAR	Les valeurs sources sont affichées sous la forme d'un diagramme à barres horizontal.
s	SCOPE	Lorsque les valeurs sources atteignent la fin de la zone d'affichage du CHART (diagramme), la zone d'affichage est effacée et les nouvelles valeurs sources sont à nouveau affichées à partir du début de la zone d'affichage.
s	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 effacées de la zone d'affichage et les nouvelles valeurs sources sont ajoutées à la zone d'affichage.
s	SWEEP	Lorsque les valeurs sources atteignent la fin de la zone d'affichage du CHART, les nouvelles valeurs sources sont à nouveau affichées au début de la zone d'affichage, mais contrairement à SCOPE, seule la partie de la zone d'affichage nécessaire pour afficher les nouvelles valeurs sources est effacée.
s	VERTICAL_BAR	Les valeurs sources sont affichées sous la forme d'un diagramme à barres vertical.

7.6 COLLECTION

7.6.1 Structure générale

Objet

Une COLLECTION est un groupe logique d'éléments, tels que des VARIABLE ou un MENU. Un identifiant est assigné à chaque élément. Les éléments peuvent être référencés dans la description du dispositif en utilisant l'identifiant de COLLECTION et le nom de l'élément.

NOTE 1 COLLECTION est destiné à être utilisé par un outil, par exemple, pour identifier des paramètres qu'il convient de traiter conjointement.

NOTE 2 Le niveau d'imbrication pris en charge recommandé est de deux (par exemple, COLLECTION de COLLECTION).

Structure lexicale

COLLECTION identifier, item-type, [MEMBERS, HELP, LABEL, VALIDITY]

Les attributs de COLLECTION sont spécifiés dans le Tableau 46.

Tableau 46 – Attributs de COLLECTION

Usage	Attribut	Description
o	item-type	Elément lexical (voir 7.6.2).
m	MEMBERS	Elément lexical (voir 7.36.10).
o	HELP	Elément lexical (voir 7.36.8).
o	LABEL	Elément lexical (voir 7.36.9).
o	VALIDITY	Elément lexical (voir 7.36.16).

7.6.2 Attribut spécifique – item-type

Objet

L'attribut item-type (type d'élément) spécifie le type de membres dans la collection. Si item-type est spécifié, tous les membres de collection doivent être du type spécifié. Sinon, les membres de collection peuvent être d'un type quelconque.

Structure lexicale

item-type

Le Tableau 47 présente les types d'option autorisés:

Tableau 47 – item-type

Usage	item-type	Description
s	AXIS	Sélection de cette construction de base.
s	BLOCK_A	Sélection de cette construction de base.
s	BLOCK_B	Sélection de cette construction de base.
s	CHART	Sélection de cette construction de base.
s	COLLECTION	Sélection de cette construction de base.
s	COMMAND	Sélection de cette construction de base.
s	COMPONENT	Sélection de cette construction de base.
s	COMPONENT_FOLDER	Sélection de cette construction de base.
s	COMPONENT_REFERENCE	Sélection de cette construction de base.
s	COMPONENT_RELATION	Sélection de cette construction de base.
s	CONNECTION	Sélection de cette construction de base.
s	DOMAIN	Sélection de cette construction de base.
s	EDIT_DISPLAY	Sélection de cette construction de base.
s	FILE	Sélection de cette construction de base.
s	GRAPH	Sélection de cette construction de base.
s	GRID	Sélection de cette construction de base.
s	IMAGE	Sélection de cette construction de base.
s	INTERFACE	Sélection de cette construction de base.
s	LIST	Sélection de cette construction de base.
s	MENU	Sélection de cette construction de base.
s	METHOD	Sélection de cette construction de base.
s	PROGRAM	Sélection de cette construction de base.
s	RECORD	Sélection de cette construction de base.
s	REFERENCE_ARRAY	Sélection de cette construction de base.
s	REFRESH	Sélection de cette construction de base.
s	RESPONSE_CODES	Sélection de cette construction de base.
s	SOURCE	Sélection de cette construction de base.
s	TEMPLATE	Sélection de cette construction de base.
s	UNIT	Sélection de cette construction de base.
s	VALUE_ARRAY	Sélection de cette construction de base.
s	VARIABLE	Sélection de cette construction de base.
s	VARIABLE_LIST	Sélection de cette construction de base.
s	WAVEFORM	Sélection de cette construction de base.
s	WRITE_AS_ONE	Sélection de cette construction de base.

7.7 COMMAND

7.7.1 Structure générale

Objet

Un objet COMMAND (commande) est une construction utilisée pour prendre en charge le mapping des éléments de communication EDD avec un système de communication choisi. Il spécifie différents éléments requis pour construire une trame de communication. Si cette construction est utilisée par un profil EDDL, chaque donnée à transmettre doit être référencée à l'intérieur d'une COMMAND et également dans le MENU de transmission/téléchargement (voir 7.23.2.9).

NOTE 1 Le champ d'adresse d'une trame de communication est spécifié en utilisant les constructions sélectionnables BLOCK, SLOT, SUB_SLOT, NUMBER, INDEX, MODULE comme requis. S'il existe des instances

de processus d'application EDD multiples, ou une instance de processus d'application EDD qui nécessite une référence explicite, celles-ci sont spécifiées par la construction additionnelle CONNECTION (connexion). Le type de trame de communication ou son champ de contrôle est spécifié par la construction OPERATION (opération). Le contenu de données de la trame de communication est spécifié par la construction TRANSACTION.

NOTE 2 Le domaine d'application de BLOCK, SLOT, SUB_SLOT, NUMBER est limité à une instance de processus d'application EDD donnée, c'est-à-dire que la même valeur de BLOCK/SLOT/NUMBER peut être utilisée conjointement avec différentes valeurs de CONNECTION pour identifier différentes données dans les différentes instances de processus de l'application EDD.

Structure lexicale

COMMAND identifier, [OPERATION, TRANSACTION+, INDEX, BLOCK_B, NUMBER, SLOT, SUB_SLOT, CONNECTION, HEADER, MODULE, RESPONSE_CODES]

Les attributs de COMMAND sont spécifiés dans le Tableau 48.

Tableau 48 – Attributs de COMMAND

Usage	Attribut	Description
m	OPERATION	Elément lexical (voir 7.7.2.1).
m	TRANSACTION	Elément lexical (voir 7.7.2.2).
c	INDEX	Elément lexical (voir 7.7.2.3).
s	BLOCK_B	Elément lexical (voir 7.7.2.4).
s	NUMBER	Elément lexical (voir 7.7.2.5).
s	SLOT	Elément lexical (voir 7.7.2.6).
s	SUB_SLOT	Elément lexical (voir 7.7.2.7).
o	CONNECTION	Elément lexical (voir 7.7.2.8).
o	HEADER	Elément lexical (voir 7.7.2.9).
o	MODULE	Elément lexical (voir 7.7.2.10).
o	RESPONSE_CODES	Elément lexical (7.36.13).

NOTE 1 Les codes de réponse apparaissant dans une transaction s'appliquent uniquement à cette transaction.

NOTE 2 Les codes de réponse apparaissant à l'extérieur d'une transaction quelconque s'appliquent à toutes les transactions.

NOTE 3 Si le même code de réponse est spécifié à la fois à l'intérieur et à l'extérieur d'une transaction, le code de réponse spécifié dans la transaction est prioritaire, mais uniquement pour cette transaction.

7.7.2 Attributs spécifiques

7.7.2.1 OPERATION

Objet

L'attribut OPERATION (opération) spécifie les actions effectuées par le dispositif de terrain lors de la réception d'une demande de service provenant du système de communication.

Structure lexicale

OPERATION [(COMMAND, DATA_EXCHANGE, READ, string, WRITE)]

Les attributs d'OPERATION sont spécifiés dans le Tableau 49.

Tableau 49 – Attributs d'OPERATION

Usage	Attribut	Description
s	COMMAND	Le dispositif effectue un ensemble prédéfini d'actions (par exemple, autodiagnostic, réinitialisation principale). Les actions à effectuer sont spécifiques au dispositif et sont identifiées par l'attribut NUMBER (numéro).

Usage	Attribut	Description
s	DATA_EXCHANGE	Spécifie une transaction bidirectionnelle. Les données d'entrée sont spécifiées dans l'attribut REPLY (réponse) et les données de sortie sont spécifiées dans l'attribut REQUEST (demande) de la TRANSACTION.
s	READ	Spécifie une transaction de lecture. Le dispositif de terrain retourne les valeurs actuelles des variables spécifiées. VARIABLES doit être défini dans la section REPLY (réponse) de TRANSACTION.
s	string	Chaîne spécifiant des services spécifiques au fabricant pour les transactions. Il convient que la transaction à exécuter soit expliquée dans une spécification correspondante.
s	WRITE	Spécifie une transaction d'écriture. Le dispositif de terrain reçoit les valeurs actuelles des variables spécifiées. VARIABLES doit être défini dans la section REQUEST (demande) de TRANSACTION.

7.7.2.2 TRANSACTION

Structure générale

Objet

Les transactions spécifient le champ de données des messages de demande et de réponse d'une commande. Il est possible de définir plus d'une transaction. Il convient que la syntaxe prenne en charge une identification unique de la TRANSACTION. Chaque TRANSACTION peut comprendre un ensemble de codes de réponse qui s'applique seulement à cette transaction particulière.

EXEMPLE Des commandes de bloc, telles que les commandes 4 et 5 de HART⁵ Universal Command version 4 ou supérieure, sont des exemples de commandes de transaction multiples.

Structure lexicale

TRANSACTION [REPLY, REQUEST, integer, RESPONSE_CODES]

Les attributs de TRANSACTION sont spécifiés dans le Tableau 50.

Tableau 50 – Attributs de TRANSACTION

Usage	Attribut	Description
m	REPLY	Élément lexical (voir 7.7.2.2.2.1).
m	REQUEST	Élément lexical (voir 7.7.2.2.2.2).
o	integer	Nombre de TRANSACTIONS (si plusieurs objets TRANSACTION sont utilisés).
o	RESPONSE_CODES	Élément lexical (voir 7.36.13).

Attributs spécifiques

7.4.2.2.2.1 REPLY

Objet

L'attribut REPLY (réponse) contient une liste de données (constantes ou références à des variables, listes, matrices, ou collections), qui sont reçues depuis le dispositif. L'ordre de la liste doit être maintenu dans le champ de données correspondant dans le PDU communiqué. Si les données référencées ne sont pas de type VARIABLE, les données référencées à l'intérieur des éléments EDD doivent toujours terminer sous forme de références de VARIABLE. Tous les autres types d'élément EDD référencés ne sont pas autorisés. La combinaison de liste, matrice, et collection avec un masque d'élément n'est pas autorisée.

⁵ Voir l'Annexe D.4.

Lorsqu'une VARIABLE ne commence et ne termine pas sur des limites d'octet, il convient qu'un masque soit utilisé pour spécifier comment la variable est encapsulée dans le champ de données.

Chaque bit dans le masque peut avoir une signification sémantique. De plus, si le bit le moins significatif est défini dans un masque d'élément, la donnée suivante (le cas échéant) est contenue dans le(s) octet(s) suivant(s) du PDU. Si le bit le moins significatif est nul (non défini), la donnée suivante est contenue dans le(s) même(s) octet(s) du PDU.

Lorsqu'une liste est référencée, l'application EDD doit réinitialiser la LIST puis doit remplir la LIST avec les données du PDU reçu.

NOTE 1 Même si le bit le moins significatif n'est pas utilisé pour des données réelles, il convient qu'il soit défini dans un masque d'élément (associé à une donnée "fictive") si des données suivent.

NOTE 2 Des exemples sont présentés dans l'Annexe C.

Structure lexicale

REPLY [reference, item-mask, INFO, INDEX]+

Les attributs de REPLY sont spécifiés dans le Tableau 51.

Tableau 51 – Attributs de REPLY et REQUEST

Usage	Attribut	Description
m	reference	Référence à une constante ou une instance de VARIABLE, ARRAY, COLLECTION, REFERENCE_ARRAY, ou LIST.
o	item-mask	Motif de bits pour extraire la VARIABLE du champ de données. Le masque d'élément doit être utilisé uniquement pour les types de données INTEGER, UNSIGNED INTEGER, ENUMERATED et BIT_ENUMERATED. Si le bit le moins significatif est défini dans le masque d'élément, l'élément suivant dans la liste REQUEST ou REPLY doit être utilisé. La longueur du masque d'élément doit être de 1 octet à 4 octets et doit correspondre à la longueur de la donnée dans la liste REQUEST ou REPLY.
o	INFO	La VARIABLE n'est pas réellement stockée dans le dispositif. Elle contient des informations additionnelles pour assurer un traitement correct.
o	INDEX	La variable est utilisée dans les objets REQUEST ou REPLY en tant qu'index dans un REFERENCE_ARRAY (matrice de référence).

NOTE 3 Une variable peut être qualifiée à la fois par INDEX et INFO auquel cas elle est appelée variable d'index local. Les commandes avec des indices locaux se comportent exactement de la même façon que les commandes avec des variables qualifiées par INDEX à l'exception du fait que les variables d'index ne sont pas stockées dans le dispositif.

NOTE 4 INFO peut être utilisé pour envoyer ou lire une VARIABLE avec des unités différentes de celles utilisées dans le dispositif.

7.4.2.2.2 REQUEST

Objet

L'attribut REQUEST (demande) contient une liste de données (constantes ou références à des variables, listes, matrices, ou collections), qui sont envoyées au dispositif. L'ordre de la liste doit être maintenu dans le champ de données correspondant dans le PDU communiqué. Si les données référencées ne sont pas de type VARIABLE, les données référencées à l'intérieur des éléments EDD doivent toujours se terminer sous forme de références de VARIABLE. Tous les autres types d'élément EDD référencés ne sont pas autorisés. La combinaison de liste, matrice, et collection avec un masque d'élément n'est pas autorisée.

Lorsqu'une donnée ne commence et ne termine pas sur des limites d'octet du PDU, un masque d'élément doit être utilisé pour spécifier comment la donnée est encapsulée dans le champ de données. Le masque d'élément est un nombre entier d'octets qui doit être logiquement soumis à une opération ET avec le PDU communiqué pour extraire la donnée souhaitée.

Si aucun masque de donnée n'est spécifié, un masque implicite est utilisé avec tous les bits définis pour chaque octet, correspondant à la longueur de donnée.

Chaque bit dans le masque peut avoir une signification sémantique. De plus, si le bit le moins significatif est défini dans un masque d'élément, la donnée suivante (le cas échéant) est contenue dans l'octet(s) suivant(s) du PDU. Si le bit le moins significatif est nul (non défini), la donnée suivante est contenue dans le(s) même(s) octet(s) du PDU.

NOTE 1 Même si le bit le moins significatif n'est pas utilisé pour des données réelles, il convient qu'il soit défini dans un masque d'élément (associé à une donnée "fictive") si des données suivent.

NOTE 2 Des exemples sont présentés dans l'Annexe C.

Structure lexicale

REQUEST integer, [reference, item-mask, INFO, INDEX]+

Les attributs de REQUEST sont spécifiés dans le Tableau 51.

7.7.2.3 INDEX

Objet

L'attribut INDEX doit être présent si l'attribut SLOT (tranche) ou BLOCK_B (bloc B) est utilisé. L'INDEX spécifie l'adresse de donnée dans le SLOT ou dans le BLOCK_B.

Structure lexicale

INDEX (integer reference, expression) <exp>

Les attributs d'INDEX sont spécifiés dans le Tableau 52.

Tableau 52 – Attributs d'INDEX

Usage	Attribut	Description
s	integer	Numéro de l'index.
s	reference	Référence à une instance de VARIABLE contenant l'index.
s	expression	Expression qui sera évaluée à un entier non négatif qui est dans l'intervalle d'index de l'attribut INDEX (pour la description des expressions, voir 7.40).

7.7.2.4 BLOCK_B

Objet

L'attribut BLOCK_B fournit une référence à un bloc individuel dans le dispositif.

Structure lexicale

BLOCK_B (reference) <exp>

L'attribut de BLOCK_B est spécifié dans le Tableau 53.

Tableau 53 – Attribut de BLOCK_B

Usage	Attribut	Description
m	reference	Référence à une instance de BLOCK_B.

7.7.2.5 NUMBER

Objet

L'attribut NUMBER (numéro) est utilisé pour énumérer les objets COMMAND (commande).

Structure lexicale

NUMBER (integer, reference, expression)<exp>

Les attributs de NUMBER sont spécifiés dans le Tableau 54.

Tableau 54 – Attributs de NUMBER

Usage	Attribut	Description
s	integer	Numéro de COMMAND.
s	reference	Référence à une instance de VARIABLE contenant le numéro de COMMAND.
s	expression	Expression qui sera évaluée à un entier non négatif qui est dans l'intervalle d'index de l'attribut NUMBER (pour la description des expressions, voir 7.40).

7.7.2.6 SLOT

Objet

L'attribut SLOT (plage) spécifie le numéro d'une plage. Les données d'un dispositif peuvent être allouées à des plages logiques. Dans une plage, chaque donnée est adressée en utilisant un index.

Structure lexicale

SLOT (integer, reference, expression)<exp>

Les attributs de SLOT sont spécifiés dans le Tableau 55.

Tableau 55 – Attributs de SLOT

Usage	Attribut	Description
s	integer	Numéro de la plage.
s	reference	Référence à une instance de VARIABLE contenant le numéro de plage.
s	expression	Expression qui sera évaluée à un entier non négatif qui est dans l'intervalle d'index de l'attribut SLOT (pour la description des expressions, voir 7.40).

7.7.2.7 SUB_SLOT

Objet

L'attribut SUB_SLOT (sous-plage) spécifie le numéro d'une sous-plage. Les données d'un dispositif peuvent être allouées à des sous-plages logiques. Dans une sous-plage, chaque donnée est adressée en utilisant un index.

Structure lexicale

SUB_SLOT (integer, reference, expression)<exp>

Les attributs de SUB_SLOT sont spécifiés dans le Tableau 56.

Tableau 56 – Attributs de SUB_SLOT

Usage	Attribut	Description
s	integer	Numéro de sous-plage.
s	reference	Référence à une instance de VARIABLE contenant le numéro de sous-plage.
s	expression	Expression qui sera évaluée à un entier non négatif qui est dans l'intervalle d'index de l'attribut SUB_SLOT (pour la description des expressions, voir 7.40).

7.7.2.8 CONNECTION

Objet

L'attribut CONNECTION (connexion) contient une référence à la construction de base CONNECTION qui contient le numéro d'une instance du processus d'application de l'EDD.

Structure lexicale

CONNECTION (reference) <exp>

L'attribut de CONNECTION est spécifié dans le Tableau 57.

Tableau 57 – Attribut de CONNECTION

Usage	Attribut	Description
m	reference	Référence à une instance de CONNECTION spécifiant l'application EDD.

7.7.2.9 HEADER

Objet

L'attribut HEADER (en-tête) est utilisé pour des protocoles de communication spécifiques.

NOTE Il peut être utilisé pour adresser des données dans un dispositif ou pour une opération de réinitialisation.

Structure lexicale

HEADER (string) <exp>

L'attribut de HEADER est spécifié dans le Tableau 58.

Tableau 58 – Attribut de HEADER

Usage	Attribut	Description
m	string	Chaîne qui est utilisée pour la communication avec des protocoles de communication spécifiques.

7.7.2.10 MODULE

Objet

L'attribut MODULE donne une référence externe. L'utilisation de cet attribut est hors du domaine d'application de la présente norme.

NOTE Le MODULE est inclus pour des raisons de compatibilité avec des dispositifs existants qui utilisent d'autres fichiers de description.

Structure lexicale

MODULE (reference) <exp>

L'attribut de MODULE est spécifié dans le Tableau 59.

Tableau 59 – Attribut de MODULE

Usage	Attribut	Description
m	reference	Référence à une instance de MODULE dans un fichier de description externe.

7.8 COMPONENT

7.8.1 Structure générale

Objet

Décrit la configuration du comportement d'un dispositif qui est décrit dans l'EDD. Un composant énumère toutes les relations avec les autres composants.

Un composant est constitué d'un fichier DDL, de fichiers d'importation, de fichiers d'inclusion, de dictionnaires, d'images, et de fichiers d'aide.

Tous les modules d'un dispositif modulaire doivent être décrits avec des objets COMPONENT (composant). Les relations entre les différents composants sont décrites avec des COMPONENT_RELATIONS (relations de composant). La hiérarchie du catalogue est décrite avec des COMPONENT_FOLDER (dossier de composant) et des COMPONENT.

Les expressions conditionnelles peuvent avoir accès à différents composants en utilisant des OBJECT_REFERENCE (référence d'objet). OBJECT_REFERENCE permet d'accéder aux variables et permet d'appeler des méthodes.

Structure lexicale

COMPONENT identifier [LABEL, CAN_DELETE, CHECK_CONFIGURATION, CLASSIFICATION, COMPONENT_PARENT, COMPONENT_PATH, COMPONENT_RELATIONS, DECLARATION, DETECT, EDD, HELP, INITIAL_VALUES, PROTOCOL, REDUNDANCY, SCAN, SCAN_LIST, SUPPLIED_INTERFACE]

Les attributs sont spécifiés dans le Tableau 60.

Tableau 60 – Attributs de COMPONENT

Usage	Attribut	Description
m	LABEL	Elément lexical (voir 7.36.9).
o	CAN_DELETE	Elément lexical (voir 7.8.2.1).
o	CHECK_CONFIGURATION	Elément lexical (voir 7.8.2.2).
o	CLASSIFICATION	Elément lexical (voir 7.36.1).
o	COMPONENT_PARENT	Elément lexical (voir 7.36.2).
o	COMPONENT_PATH	Elément lexical (voir 7.36.3).
o	COMPONENT_RELATIONS	Elément lexical (voir 7.8.2.3).
o	DECLARATION	Elément lexical (voir 7.8.2.4).
o	DETECT	Elément lexical (voir 7.8.2.5).
o	EDD	Elément lexical (voir 7.8.2.6).
o	HELP	Elément lexical (voir 7.36.8).
o	INITIAL_VALUES	Elément lexical (voir 7.8.2.7).
o	PROTOCOL	Elément lexical (voir 7.36.13).
o	REDUNDANCY	Elément lexical (voir 7.8.2.8).
o	SCAN	Elément lexical (voir 7.8.2.9).
o	SCAN_LIST	Elément lexical (voir 7.8.2.10).
o	SUPPLIED_INTERFACE	Elément lexical (voir 7.36.15).

7.8.2 Attributs spécifiques

7.8.2.1 CAN_DELETE

Objet

L'attribut CAN_DELETE (suppression possible) est égal à TRUE (vrai) ou FALSE (faux) et indique s'il est possible de supprimer sans danger l'instance du module en question d'un dispositif modulaire. Cet attribut est utilisé dans des cas d'utilisation en ligne et hors ligne. Par défaut, CAN_DELETE est TRUE. Un exemple de son utilisation est: les canaux d'un module ne peuvent pas être supprimés; seul le module dans son ensemble peut être supprimé.

Structure lexicale

CAN_DELETE (FALSE, TRUE) <exp>

Les attributs de CAN_DELETE sont spécifiés dans le Tableau 61.

Tableau 61 – Attributs de CAN_DELETE

Usage	Attribut	Description
s	TRUE	Le composant peut être supprimé.
s	FALSE	Le composant ne peut pas être supprimé.

7.8.2.2 CHECK_CONFIGURATION

Objet

L'attribut CHECK_CONFIGURATION (vérifier configuration) fait référence à une méthode qui doit être appelée par l'application EDD pour confirmer la configuration actuelle de ce COMPONENT. Une valeur de retour BI_SUCCESS indique que la configuration est valide; toutes les autres valeurs indiquent que la configuration est non valide. Si la méthode échoue, la configuration est alors non valide. La méthode référencée ne doit avoir aucun paramètre d'entrée et un paramètre de sortie unique de type DD_STRING qui retourne un message. Ce message doit être affiché à l'utilisateur par l'application EDD. Ce message peut contenir plusieurs explications à une configuration non valide. Dans ce cas, le message peut contenir des sauts de paragraphe et des sauts de ligne. Cette méthode peut être utilisée en ligne et hors ligne.

Structure lexicale

CHECK_CONFIGURATION (NULL, string) <exp>

Les attributs de CHECK_CONFIGURATION sont spécifiés dans le Tableau 62.

Tableau 62 – Attributs de CHECK_CONFIGURATION

Usage	Attribut	Description
s	NULL	La configuration est correcte.
s	string	Chaîne typique générée par une METHOD. Cette chaîne comprend la description du problème et peut être une suggestion pour résoudre le problème.

7.8.2.3 COMPONENT_RELATIONS

Objet

L'attribut COMPONENT_RELATIONS (relations de composant) répertorie les relations avec d'autres composants que le dispositif modulaire ou module en question peut prendre en

charge. Cette EDD peut répertorier des modules apparentés, des modules enfants et des modules parents.

Structure lexicale

COMPONENT_RELATIONS [(component-relation-reference)<exp>]+

L'attribut de COMPONENT_RELATIONS est spécifié dans le Tableau 63.

Tableau 63 – Attribut de COMPONENT_RELATIONS

Usage	Attribut	Description
m	component-relation-reference	Liste d'identifiants de relation de composant. Des expressions conditionnelles sont autorisées.

7.8.2.4 DECLARATION

Objet

L'attribut DECLARATION (déclaration) spécifie des constructions de base associées au composant.

Structure lexicale

DECLARATION [item]+

L'attribut de DECLARATION est spécifié dans le Tableau 64.

Tableau 64 – Attribut de DECLARATION

Usage	Attribut	Description
m	item	Déclarations de constructions de base (voir 7.1.4).

7.8.2.5 DETECT

Objet

L'attribut DETECT (détection) indique la méthode que l'application EDD doit appeler pour détecter un composant.

NOTE Dans HART, il convient que l'application EDD s'assure de lire avec la commande 75 pour identifier un composant.

Structure lexicale

DETECT method_reference

L'attribut de DETECT est spécifié dans le Tableau 65.

Tableau 65 – Attribut de DETECT

Usage	Attribut	Description
s	method_reference	Référence à une méthode de détection.

7.8.2.6 EDD

Objet

L'attribut EDD spécifie un fichier qui contient le reste de l'EDD. Celui-ci est requis uniquement si la description de composant est dans un fichier séparé.

Structure lexicale

EDD string

L'attribut d'EDD est spécifié dans le Tableau 66.

Tableau 66 – Attribut d'EDD

Usage	Attribut	Description
m	string	Chaîne indiquant le fichier qui contient le reste de l'EDD.

7.8.2.7 INITIAL_VALUES

Objet

L'attribut INITIAL_VALUES (valeurs initiales) est une liste d'initialisation pour des variables. Même si les variables ont une INITIAL_VALUE (valeur initiale) ou une DEFAULT_VALUE (valeur par défaut), ces INITIAL_VALUES sont utilisées pour l'initialisation.

Elles peuvent être utilisées, par exemple, pour définir différentes initialisations pour différents usages, par exemple la mesure de pression différentielle, de niveau et de débit pour un transducteur de pression.

Structure lexicale

INITIAL_VALUES [reference, value<exp>] +

Les attributs de CHECK_CONFIGURATION sont spécifiés dans le Tableau 62.

Tableau 67 – Attributs de CHECK_CONFIGURATION

Usage	Attribut	Description
s	reference	Référence à une variable.
s	value	Valeur initiale de la variable référencée.

7.8.2.8 REDUNDANCY

Objet

L'attribut REDUNDANCY (redondance) contient un entier (par défaut égal à 1) qui indique combien de connecteurs redondants sont pris en charge par le dispositif en question.

Structure lexicale

REDUNDANCY (integer) <exp>

Les attributs de REDUNDANCY sont spécifiés dans le Tableau 68.

Tableau 68 – Attribut de REDUNDANCY

Usage	Attribut	Description
o	integer	Nombre entier de connecteurs Fieldbus.

7.8.2.9 SCAN

Objet

L'attribut SCAN (analyse) indique la méthode que l'application EDD doit appeler pour créer une liste d'actions d'un dispositif modulaire.

NOTE Dans HART, il convient que l'application EDD lise avec la commande 74 la liste des modules existants.

Structure lexicale

SCAN method_reference

L'attribut de SCAN est spécifié dans le Tableau 69.

Tableau 69 – Attribut de SCAN

Usage	Attribut	Description
s	method_reference	Référence à la méthode d'analyse.

7.8.2.10 SCAN_LIST

Objet

L'attribut SCAN_LIST (liste d'analyse) est une référence à la liste de résultats de sous-composants qui est créée par la méthode d'analyse.

NOTE Dans HART, il convient que l'application EDD lise avec la commande 74 la liste des modules existants.

Structure lexicale

SCAN_LIST reference

L'attribut de SCAN_LIST est spécifié dans le Tableau 70.

Tableau 70 – Attribut de SCAN_LIST

Usage	Attribut	Description
s	reference	Référence à la liste d'analyse.

7.9 COMPONENT_FOLDER

Objet

Un COMPONENT_FOLDER (dossier de composant) décrit un dossier dans un catalogue. Les dossiers peuvent comprendre d'autres dossiers. Les dossiers permettent de grouper des composants, par exemple les composants d'une famille de produits. Les LABEL (libellé) des dossiers sont associés au COMPONENT_PATH (chemin de composant).

Structure lexicale

COMPONENT_FOLDER _identifiant [LABEL, CLASSIFICATION, COMPONENT_PARENT, COMPONENT_PATH, HELP, PROTOCOL]

Les attributs de COMPONENT_FOLDER sont spécifiés dans le Tableau 71.

Tableau 71 – Attributs de COMPONENT_FOLDER

Usage	Attribut	Description
m	LABEL	Élément lexical (voir 7.36.9).
o	CLASSIFICATION	Élément lexical (voir 7.36.1).
o	COMPONENT_PARENT	Élément lexical (voir 7.36.2).
o	COMPONENT_PATH	Élément lexical (voir 7.36.3).
o	HELP	Élément lexical (voir 7.36.8).
o	PROTOCOL	Élément lexical (voir 7.36.15).

7.10 COMPONENT_REFERENCE

Objet

Un COMPONENT_REFERENCE (référence de composant) désigne un autre composant via l'attribut COMPONENT_PARENT (composant parent) ou avec les attributs PROTOCOL (protocole), CLASSIFICATION, MANUFACTURER (fabricant) et COMPONENT_PATH (chemin de composant). Si un dossier est référencé, tous les composants qui sont inclus dans ce dossier sont référencés.

Si le COMPONENT_REFERENCE correspond à un dossier contenant une EDD unique, le COMPONENT_REFERENCE concerne un dispositif unique. Sinon, une famille de dispositifs est indiquée.

En variante à une CLASSIFICATION, il est également possible de référencer un composant avec les attributs DEVICE_TYPE (type de dispositif) et DEVICE_REVISION (révision de dispositif).

Structure lexicale

COMPONENT_REFERENCE identifier [CLASSIFICATION, COMPONENT_PARENT, COMPONENT_PATH, DEVICE_REVISION, DEVICE_TYPE, MANUFACTURER, PROTOCOL]

Les attributs de COMPONENT_FOLDER sont spécifiés dans le Tableau 72.

Tableau 72 – Attribut de COMPONENT_REFERENCE

Usage	Attribut	Description
o	CLASSIFICATION	Élément lexical (voir 7.36.1).
o	COMPONENT_PARENT	Élément lexical (voir 7.36.2).
o	COMPONENT_PATH	Élément lexical (voir 7.36.3).
o	DEVICE_REVISION	Élément lexical (voir 7.2.2.2).
o	DEVICE_TYPE	Élément lexical (voir 7.2.2.3).
o	MANUFACTURER	Élément lexical (voir 7.2.2.6), par défaut le MANUFACTURER (fabricant) de l'EDD.
o	PROTOCOL	Élément lexical (voir 7.36.13).

7.11 COMPONENT_RELATION

7.11.1 Structure générale

Objet

Un COMPONENT_RELATION (relation de composant) spécifie une relation entre un module et un ou plusieurs autres modules et les contraintes qui doivent être remplies pour instancier ces autres modules.

Structure lexicale

COMPONENT_RELATION identifier [COMPONENTS, LABEL, RELATION_TYPE, ADDRESSING, HELP, MAXIMUM_NUMBER, MINIMUM_NUMBER, REQUIRED_INTERFACE, SUPPLIED_INTERFACE]

Les attributs de COMPONENT_RELATION sont spécifiés dans le Tableau 73.

Tableau 73 – Attributs de COMPONENT_RELATION

Usage	Attribut	Description
m	COMPONENTS	Elément lexical (voir 7.11.2.1).
m	LABEL	Elément lexical (voir 7.36.9).
m	RELATION_TYPE	Elément lexical (voir 7.11.2.2).
o	ADDRESSING	Elément lexical (voir 7.11.2.3).
o	HELP	Elément lexical (voir 7.36.8).
o	MAXIMUM_NUMBER	Elément lexical (voir 7.11.2.4).
o	MINIMUM_NUMBER	Elément lexical (voir 7.11.2.5).
o	REQUIRED_INTERFACE	Elément lexical (voir 7.11.2.6).
o	SUPPLIED_INTERFACE	Elément lexical (voir 7.36.15).

7.11.2 Attributs spécifiques

7.11.2.1 COMPONENTS

Objet

L'attribut COMPONENTS (composants) répertorie les modules ou les dispositifs modulaires autorisés et les contraintes liées à leur utilisation.

Structure lexicale

COMPONENTS [component-reference<exp> AUTO_CREATE, MINIMUM_NUMBER, MAXIMUM_NUMBER, FILTER, REQUIRED_RANGES]+

Les attributs de COMPONENTS sont spécifiés dans le Tableau 74.

Tableau 74 – Attributs de COMPONENTS

Usage	Attribut	Description
m	composant-reference	Composant dans la relation.
o	AUTO_CREATE	Objet Nombre de composants automatiquement créés. Structure lexicale AUTO_CREATE (expression)<exp>
o	FILTER	Objet Utilisé si la référence désigne un COMPONENT_FOLDER (dossier de composants) et spécifie une expression booléenne qui permet de filtrer les composants. Des composants sont inclus si l'expression est TRUE (vrai). Pour filtrer des composants spéciaux, des références au type de composant sont utilisées dans l'expression. La condition est évaluée pour chaque type contenu dans le COMPONENT_FOLDER. Structure lexicale FILTER (TRUE, FALSE)<exp>
o	MAXIMUM_NUMBER	Objet Nombre maximal autorisé de ce composant. Structure lexicale MAXIMUM_NUMBER (expression)<exp>
o	MINIMUM_NUMBER	Objet Nombre minimal requis de ce composant. Structure lexicale MINIMUM_NUMBER (expression)<exp>
o	REQUIRED_RANGES	Objet Variables avec intervalles. Dans une configuration valide, les variables doivent être dans l'intervalle spécifié. Cela est typiquement utilisé pour définir des restrictions pour l'adressage (par exemple, une page).

Usage	Attribut	Description
		Structure lexicale REQUIRED_RANGES [reference, [MAX_VALUE, m]*, [MIN_VALUE, n]*]+

7.11.2.2 RELATION_TYPE

Objet

L'attribut RELATION_TYPE (type de relation) définit la relation entre le composant en question et d'autres composants (enfants, parents ou apparentés).

Structure lexicale

RELATION_TYPE [CHILD_COMPONENT, PARENT_COMPONENT, SIBLING_COMPONENT]

Les attributs de RELATION_TYPE sont spécifiés dans le Tableau 75.

Tableau 75 – Attributs de RELATION_TYPE

Usage	Attribut	Description
s	CHILD_COMPONENT	Liste de composants qui sont des sous-modules autorisés.
s	PARENT_COMPONENT	Liste de composants qui sont des modules ou des dispositifs modulaires parents autorisés pour ce composant.
s	SIBLING_COMPONENT	Relation entre des composants apparentés qui peut requérir ou exclure d'autres composants. Il est possible de requérir d'autre composant quand MINIMUM_NUMBER > 0. L'exclusion est possible avec MAXIMUM_NUMBER = 0.
s	NEXT_COMPONENT	Relation de parenté spéciale avec le composant suivant associée à l'attribut ADDRESSING (adressage). L'utilisation de NEXT_COMPONENT est autorisée uniquement si ADDRESSING est spécifié. Non applicable pour HART.
s	PREV_COMPONENT	Relation de parenté spéciale avec le composant précédent associée à l'attribut ADDRESSING (adressage). L'utilisation de PREV_COMPONENT (composant précédent) n'est autorisée que si ADDRESSING est spécifié. Non applicable for HART.

7.11.2.3 ADDRESSING

Objet

L'attribut ADDRESSING (adressage) spécifie une liste de variables qui adresse un composant de façon unique.

Structure lexicale

ADDRESSING variable-reference+

L'attribut d'ADDRESSING est spécifié dans le Tableau 76.

Tableau 76 – Attribut d'ADDRESSING

Usage	Attribut	Description
m	variable-reference	Référence à une variable.

7.11.2.4 MAXIMUM_NUMBER

Objet

L'attribut MAXIMUM_NUMBER (nombre maximal) spécifie le nombre de composants qui sont autorisés. La valeur par défaut de MAXIMUM_NUMBER est 1.

Structure lexicale

MAXIMUM_NUMBER (integer)<exp>

L'attribut de MAXIMUM_NUMBER est spécifié dans le Tableau 77.

Tableau 77 – Attribut de MAXIMUM_NUMBER

Usage	Attribut	Description
o	integer	Nombre total maximal autorisé de l'ensemble des composants répertoriés.

7.11.2.5 MINIMUM_NUMBER**Objet**

L'attribut MINIMUM_NUMBER (nombre minimal) spécifie le nombre de composants qui sont requis. La valeur par défaut de MINIMUM_NUMBER est 0.

Structure lexicale

MINIMUM_NUMBER (integer)<exp>

L'attribut de MINIMUM_NUMBER est spécifié dans le Tableau 78.

Tableau 78 – Attribut de MINIMUM_NUMBER

Usage	Attribut	Description
o	integer	Nombre total minimal requis de l'ensemble des composants répertoriés.

7.11.2.6 REQUIRED_INTERFACE**Objet**

L'attribut REQUIRED_INTERFACE répertorie les interfaces requises du composant associé.

Structure lexicale

REQUIRED_INTERFACE interface-reference+

L'attribut de REQUIRED_INTERFACE est spécifié dans le Tableau 79.

Tableau 79 – Attribut de REQUIRED_INTERFACE

Usage	Attribut	Description
m	interface-reference	Référence à une interface.

7.12 CONNECTION**7.12.1 Structure générale****Objet**

Un dispositif peut prendre en compte plusieurs applications EDD. Chaque application EDD est représentée sous la forme d'une instance de processus d'application EDD. La construction de base CONNECTION (connexion) fournit une référence à une instance de processus d'application EDD d'un dispositif.

Le nom associé à une construction CONNECTION peut être utilisé dans l'attribut CONNECTION de COMMAND.

Structure lexicale

CONNECTION *identifier*, APPINSTANCE

L'attribut de CONNECTION est spécifié dans le Tableau 80.

Tableau 80 – Attribut de CONNECTION

Usage	Attribut	Description
m	APPINSTANCE	Élément lexical (voir 7.12.2).

7.12.2 Attribut spécifique – APPINSTANCE

Objet

L'attribut APPINSTANCE (instance d'application) spécifie une instance de processus de l'application EDD. Il est possible de définir plusieurs applications EDD dans un dispositif. Chaque application EDD représente une instance de processus de l'application EDD. Les instances de processus de l'application EDD permettent différents niveaux d'accès.

Structure lexicale

APPINSTANCE (*integer*, *reference*)<exp>

Les attributs d'APPINSTANCE sont spécifiés dans le Tableau 81.

Tableau 81 – Attributs d'APPINSTANCE

Usage	Attribut	Description
s	integer	Numéro de l'instance du processus de l'application EDD.
s	reference	Référence à une instance de VARIABLE contenant le nombre d'instances de processus d'application EDD.

7.13 DOMAIN

Objet

Un DOMAIN (domaine) représente un espace mémoire dans un dispositif. Il peut contenir des programmes ou des données. L'identifiant spécifie le DOMAIN correspondant du dispositif. Les données, programmes (qui sont alloués dans le DOMAIN), taille, accès, etc., sont spécifiés à l'extérieur de l'EDD.

NOTE 1 DOMAIN peut être utilisé pour transférer de grandes quantités de données vers et depuis un dispositif.

NOTE 2 Afin de permettre la transmission ou le téléchargement de DOMAIN, il est nécessaire que le dispositif prenne en charge les services spécifiques au DOMAIN.

Structure lexicale

DOMAIN *identifier* [HANDLING, RESPONSE_CODES]

Les attributs de DOMAIN sont spécifiés dans le Tableau 82.

Tableau 82 – Attributs de DOMAIN

Usage	Attribut	Description
o	HANDLING	Élément lexical (voir 7.36.7).
o	RESPONSE_CODES	Élément lexical (voir 7.36.13).

7.14 EDIT_DISPLAY

7.14.1 Structure générale

Objet

Un élément EDIT_DISPLAY (édition-affichage) définit comment les données seront gérées pour l'affichage et l'édition. Il est utilisé pour grouper des éléments conjointement pour édition.

NOTE Cette construction est incluse pour la rétrocompatibilité. Pour de futures implémentations, la construction de base plus générale de MENU est recommandée.

Structure lexicale

EDIT_DISPLAY identifier [EDIT_ITEMS, LABEL, DISPLAY_ITEMS, PRE_EDIT_ACTIONS, POST_EDIT_ACTIONS]

Les attributs d'EDIT_DISPLAY sont spécifiés dans le Tableau 83.

Tableau 83 – Attributs de EDIT_DISPLAY

Usage	Attribut	Description
m	EDIT_ITEMS	Élément lexical (voir 7.14.2.1).
m	LABEL	Élément lexical (voir 7.36.9).
o	DISPLAY_ITEMS	Élément lexical (voir 7.14.2.2).
o	POST_EDIT_ACTIONS	Élément lexical (voir 7.14.2.3).
o	PRE_EDIT_ACTIONS	Élément lexical (voir 7.14.2.4).

7.14.2 Attributs spécifiques

7.14.2.1 EDIT_ITEMS

Objet

L'attribut EDIT_ITEMS (éléments pour édition) définit l'ensemble des données qui doivent être présentées à l'utilisateur, et peuvent être éditées par l'utilisateur: VARIABLES, relations WRITE_AS_ONE, paramètres de BLOCK_A, et éléments de paramètres de BLOCK_A (c'est-à-dire, les champs ou éléments de PARAMETERS de BLOCK_A de type RECORD (enregistrement) ou VALUE_ARRAY (matrice de valeurs)).

NOTE 1 Les détails spécifiques pour la présentation et l'édition d'une donnée (par exemple, valeur par défaut, facteur d'échelle, intervalle), sont spécifiés dans les définitions de chaque élément individuel référencé.

NOTE 2 Si une relation WRITE_AS_ONE apparaît en tant qu'élément EDIT_ITEM, il convient que l'utilisateur examine toutes les variables dans la relation WRITE_AS_ONE. Il n'est pas nécessaire que l'utilisateur modifie toutes les valeurs mais il convient au moins qu'il valide que les valeurs actuelles sont acceptables.

Structure lexicale

EDIT_ITEMS [(reference)<exp>]+

L'attribut d'EDIT_ITEMS est spécifié dans le Tableau 84.

Tableau 84 – Attribut d'EDIT_ITEMS

Usage	Attribut	Description
m	reference	Référence à une instance de VARIABLE, WRITE_AS_ONE, BLOCK_A PARAMETERS ou un élément de BLOCK_A PARAMETERS. Les références autorisées sont spécifiques au profil.

7.14.2.2 DISPLAY_ITEMS

Objet

L'attribut DISPLAY_ITEMS (éléments pour affichage) définit l'ensemble des données qui doivent être présentées à l'utilisateur pour information, c'est-à-dire que celles-ci ne doivent pas être éditées: VARIABLES, BLOCK_A PARAMETERS, et éléments de BLOCK_A PARAMETERS (c'est-à-dire, respectivement des champs ou des éléments de BLOCK_A PARAMETERS de type RECORD ou VALUE_ARRAY).

NOTE Les détails spécifiques pour la présentation et l'édition d'une donnée (par exemple, valeur par défaut, facteur d'échelle, intervalle) sont définis dans les définitions de chaque élément individuel référencé.

Structure lexicale

DISPLAY_ITEMS [(reference)<exp>]+

L'attribut de DISPLAY_ITEMS est spécifié dans le Tableau 85.

Tableau 85 – Attribut de DISPLAY_ITEM

Usage	Attribut	Description
m	reference	Référence à une instance de VARIABLE, WRITE_AS_ONE, BLOCK_A PARAMETERS ou un élément de BLOCK_A PARAMETERS. Les références autorisées sont spécifiques au profil.

7.14.2.3 POST_EDIT_ACTIONS

Objet

L'attribut POST_EDIT_ACTIONS (actions post-édition) spécifie les METHOD qui doivent être exécutées lorsque l'utilisateur a fini le traitement de l'objet EDIT_DISPLAY (édition/affichage). Si l'EDIT_DISPLAY est abandonné, les POST_EDIT_ACTIONS ne sont pas exécutées. Les METHOD spécifiées doivent être exécutées dans l'ordre dans lequel elles apparaissent. Si une méthode se termine anormalement, les METHOD suivantes ne sont pas exécutées.

Les POST_EDIT_ACTIONS éventuelles d'une entité référencée dans les EDIT_ITEMS (éléments pour édition) doivent être exécutées lorsqu'une nouvelle valeur a été entrée. Les POST_EDIT_ACTIONS d'un élément EDIT_DISPLAY sont exécutées uniquement après que toutes ces POST_EDIT_ACTIONS individuelles soient terminées.

L'attribut POST_EDIT_ACTIONS ne doit pas être exprimé en utilisant des instructions conditionnelles IF, IF-ELSE, SELECT. Si un comportement conditionnel est requis, les méthodes appelées par l'action doivent être compatibles avec celui-ci.

Structure lexicale

POST_EDIT_ACTIONS [reference]+, DEFINITION

Les attributs de POST_EDIT_ACTIONS sont spécifiés dans le Tableau 86.

7.14.2.4 PRE_EDIT_ACTIONS

Objet

L'attribut PRE_EDIT_ACTIONS (actions pré-édition) spécifie les METHOD qui doivent être exécutées immédiatement lorsque l'EDIT_DISPLAY est activé. Les METHOD spécifiées doivent être exécutées dans l'ordre dans lequel elles apparaissent. Si une méthode se termine anormalement, les METHOD suivantes ne sont pas exécutées et l'activation d'EDIT_DISPLAY est annulée.

Les PRE_EDIT_ACTIONS de l'objet EDIT_DISPLAY doivent être exécutées avant toute PRE_EDIT_ACTIONS définie des entités référencées.

L'attribut PRE_EDIT_ACTIONS ne doit pas être exprimé en utilisant une instruction conditionnelle (IF, IF-ELSE, SELECT). Si un comportement conditionnel est requis, les méthodes appelées par l'action doivent être compatibles avec celui-ci.

NOTE Les PRE_EDIT_ACTIONS de tous les objets VARIABLE associés sont exécutés immédiatement avant que l'utilisateur commence effectivement à les éditer, par exemple, en plaçant un curseur dans un champ d'édition. L'affichage d'objets VARIABLE sans édition ne déclenche pas les PRE_EDIT_ACTIONS des objets VARIABLE.

Structure lexicale

PRE_EDIT_ACTIONS [reference]+, DEFINITION

Les attributs de PRE_EDIT_ACTIONS sont spécifiés dans le Tableau 86.

Tableau 86 – Attributs de POST_EDIT_ACTIONS, PRE_EDIT_ACTIONS

Usage	Attribut	Description
m	reference	Référence à une instance de METHOD ou une instance de METHOD avec des arguments.
o	DEFINITION	Élément lexical (voir 7.36.1).

7.15 FILE

Objet

FILE (fichier) décrit une zone de stockage persistant qui est gérée par l'application EDD. La zone mémoire doit contenir uniquement des données. Chaque dispositif physique a sa propre copie de la zone de stockage, c'est-à-dire un élément FILE associé à une instance d'un dispositif et non au type d'un dispositif. L'attribut MEMBERS (membres) définit la structure de la zone mémoire, qui doit être fixe, ce qui signifie que l'attribut MEMBERS ne doit pas être exprimé en utilisant une instruction conditionnelle IF, IF-ELSE, SELECT.

Structure lexicale

FILE identifier [MEMBERS, HELP, LABEL]

Les attributs de FILE sont spécifiés dans le Tableau 87.

Tableau 87 – Attributs de FILE

Usage	Attribut	Description
m	MEMBERS	Élément lexical (voir 7.36.12).
o	HELP	Élément lexical (voir 7.36.8).
o	LABEL	Élément lexical (voir 7.36.9).

7.16 GRAPH

7.16.1 Structure générale

Objet

GRAPH (graphique) décrit un graphique utilisé pour afficher des données provenant d'un dispositif de l'application EDD. Un GRAPH est utilisé pour afficher un ensemble de données qui est stocké dans un dispositif, par opposition à un CHART (diagramme) qui est utilisé pour afficher des valeurs de données collectées en continu depuis un dispositif.

Structure lexicale

GRAPH identifier [MEMBERS, CYCLE_TIME, HEIGHT, HELP, LABEL, VALIDITY, WIDTH, X_AXIS]

Les attributs de GRAPH sont spécifiés dans le Tableau 88.

Tableau 88 – Attributs de GRAPH

Usage	Attribut	Description
m	MEMBERS	Élément lexical (voir 7.36.12).
o	CYCLE_TIME	Élément lexical (voir 7.16.2.1).
o	HEIGHT	Élément lexical (voir 7.36.5).
o	HELP	Élément lexical (voir 7.36.8).
o	LABEL	Élément lexical (voir 7.36.9).
o	VALIDITY	Élément lexical (voir 7.38.16).
o	WIDTH	Élément lexical (voir 7.36.15).
o	X_AXIS	Élément lexical (voir 7.16.2.2).

7.16.2 Attributs spécifiques

7.16.2.1 CYCLE_TIME

Objet

L'attribut CYCLE_TIME (durée de cycle) spécifie le temps, en ms, durant lequel l'application EDD doit attendre entre des lectures successives des valeurs sources du dispositif. Par défaut, un GRAPH n'a pas de durée de cycle.

Structure lexicale

CYCLE_TIME (integer)<exp>

L'attribut de CYCLE_TIME est spécifié dans le Tableau 89.

Tableau 89 – Attribut de CYCLE_TIME

Usage	Attribut	Description
m	integer	Temps, en ms, entre des lectures successives des valeurs sources du dispositif.

7.16.2.2 X_AXIS

Objet

L'attribut X_AXIS spécifie le mode d'affichage de l'axe X. Chaque WAVEFORM (courbe) sur un GRAPH partage un X_AXIS commun mais chaque WAVEFORM sur un GRAPH a son propre Y_AXIS (axe Y). Par défaut, l'affichage de l'axe X d'un GRAPH sans attribut X_AXIS est dérivé des WAVEFORM sur le GRAPH.

Structure lexicale

X_AXIS (reference)<exp>

L'attribut de X_AXIS est spécifié dans le Tableau 90.

Tableau 90 – Attribut de X_AXIS

Usage	Attribut	Description
m	reference	Référence à une instance d'AXIS.

7.17 GRID

7.17.1 Structure générale

Objet

GRID (grille) décrit une matrice de données provenant d'un dispositif 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.

Structure lexicale

GRID identifier [VECTORS, HANDLING, HEIGHT, HELP, LABEL, ORIENTATION, VALIDITY, WIDTH]

Les attributs de GRID sont spécifiés dans le Tableau 91.

Tableau 91 – Attributs de GRID

Usage	Attribut	Description
m	VECTORS	Élément lexical (voir 7.17.2.1).
o	HANDLING	Élément lexical (voir 7.36.6).
o	HEIGHT	Élément lexical (voir 7.36.5).
o	HELP	Élément lexical (voir 7.36.8).
o	LABEL	Élément lexical (voir 7.36.9).
o	ORIENTATION	Élément lexical (voir 7.17.2.2).
o	VALIDITY	Élément lexical (voir 7.36.16).
o	WIDTH	Élément lexical (voir 7.36.17).

7.17.2 Attributs spécifiques

7.17.2.1 VECTORS

Objet

L'attribut VECTORS (vecteurs) spécifie le contenu de l'élément GRID. Chaque vecteur est affiché dans l'ordre. Pour une GRID verticale, les vecteurs sont affichés de gauche à droite. Pour une GRID horizontale, les vecteurs sont affichés de haut en bas. Chaque vecteur est constitué d'une description qui constitue une étiquette pour le vecteur suivi par les données à afficher.

Structure lexicale

VECTORS [description, (reference, string, double, float, integer, unsigned_integer)<exp>+]

Les attributs de VECTORS sont spécifiés dans le Tableau 92.

Tableau 92 – Attribut de VECTORS

Usage	Attribut	Description
m	description	Chaîne contenant une brève description des données contenues dans le vecteur.
o	reference	Référence à une instance de VARIABLE, un membre d'une instance de RECORD, un élément d'une instance de VALUE_ARRAY, un élément d'une instance de REFERENCE_ARRAY, une instance de VALUE_ARRAY ou une instance de REFERENCE_ARRAY.
	string	Chaîne constante ou référence représentée par une chaîne.
o	double	Constante à virgule flottante à double précision.
o	float	Constante à virgule flottante à simple précision.
o	integer	Constante entière.
o	unsigned_integer	Constante entière non signée.

7.17.2.2 ORIENTATION

Objet

L'attribut ORIENTATION spécifie la direction selon laquelle les vecteurs sont affichés. Par défaut, une GRID sans attribut ORIENTATION est affichée verticalement.

Structure lexicale

ORIENTATION (HORIZONTAL, VERTICAL) <exp>

Les attributs d'ORIENTATION sont spécifiés dans le Tableau 93.

Tableau 93 – Attribut d'ORIENTATION

Usage	Attribut	Description
s	HORIZONTAL	Les vecteurs sont affichés en lignes.
s	VERTICAL	Les vecteurs sont affichés en colonnes.

7.18 IMAGE

7.18.1 Structure générale

Objet

IMAGE décrit une image visuelle.

Structure lexicale

IMAGE identifiant [PATH, HELP, LABEL, LINK, VALIDITY]

Les attributs d'IMAGE sont spécifiés dans le Tableau 94.

Tableau 94 – Attributs d'IMAGE

Usage	Attribut	Description
m	PATH	Élément lexical (voir 7.18.2.1).
o	HELP	Élément lexical (voir 7.36.8).
o	LABEL	Élément lexical (voir 7.36.9).
o	LINK	Élément lexical (voir 7.18.2.2).
o	VALIDITY	Élément lexical (voir 7.36.167.36.16).

7.18.2 Attributs spécifiques

7.18.2.1 PATH

Objet

L'attribut PATH spécifie le nom du fichier de l'image. Des images localisés multiples peuvent être spécifiées en utilisant la même technique de localisation que celle utilisée avec les chaînes de caractères. Le code pays ZZ peut être utilisé pour spécifier une version basse résolution de l'image, ceci est utile pour les applications EDD qui fonctionnent sur des dispositifs mobiles ayant une mémoire restreinte.

EXAMPLE "en|english.jpeg|de|german.jpeg|en ZZ|english-lowres.jpeg|de ZZ|german-lowres.jpeg"

Structure lexicale

PATH (file-name)<exp>

L'attribut de PATH est spécifié dans le Tableau 95.

Tableau 95 – Attribut de PATH

Usage	Attribut	Description
m	file-name	Littéral chaîne qui identifie le nom de fichier de l'image.

7.18.2.2 LINK

Objet

L'attribut LINK spécifie un MENU ou une METHOD qui peut être accessible depuis l'IMAGE. Cela permet d'obtenir des informations additionnelles concernant l'IMAGE.

Structure lexicale

LINK (reference)<exp>

L'attribut de LINK est spécifié dans le Tableau 96.

Tableau 96 – Attribut de LINK

Usage	Attribut	Description
m	reference	Référence à une instance de MENU ou une instance de METHOD.

7.19 IMPORT

7.19.1 Structure générale

Objet

Les constructions de langage EDDL précédemment spécifiées dans cet article sont suffisantes pour décrire un dispositif individuel quelconque. Cependant, des mécanismes additionnels sont nécessaires pour décrire des révisions multiples d'un dispositif individuel ou décrire des parties d'un dispositif générique. Pour produire ces mécanismes, une EDD peut importer une autre EDD. L'EDD importée peut elle-même importer également d'autres EDD.

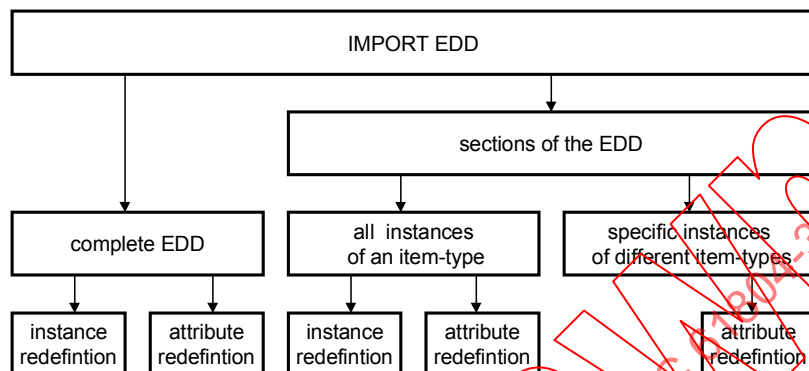
NOTE Avec ces mécanismes, l'EDD d'une nouvelle version du dispositif peut être spécifiée en important simplement l'EDD de l'ancienne version du dispositif et en spécifiant les modifications d'éléments. Ce type d'EDD est appelé description delta, car l'EDD entière est spécifiée par des modifications d'une EDD existante.

Trois types d'importation doivent être pris en charge:

- importation de tous les éléments, où tous les éléments de l'EDD externe sont importés;

- importation des éléments d'un type spécifié, où uniquement les éléments des types spécifiés sont importés. Si un type spécifié est importé, des importations d'éléments spécifiques du même type ne sont pas autorisés;
- importation d'un élément spécifique.

L'identifiant d'un élément EDD importé ne doit pas être modifié. La Figure 27 présente les mécanismes d'importation EDDL.



IEC 1536/06

Légende

Anglais	Français
IMPORT EDD	IMPORT EDD
sections of the EDD	sections de l'EDD
complete EDD	EDD entière
all instances of an item-type	toutes instances d'un élément-type
specific instances of different item-types	instances spécifiques de différents élément-type
instance redefinition	redéfinition d'instance
attribute redefinition	redéfinition d'attribut

Figure 27 – Mécanismes d'importation EDDL

Une autre méthode d'importation consiste à référencer l'EDD par le nom du fichier.

Structure lexicale

```

IMPORT ((DE_REVISION, DEVICE_REVISION, DEVICE_TYPE, EDD_PROFILE,
EDD_VERSION, MANUFACTURER, MANUFACTURER_EXT), FILE_NAME), (EVERYTHING,
(AXES, BLOCKS_A, BLOCKS_B, CHART, COLLECTIONS, COMMANDS, COMPONENTS,
COMPONENT_FOLDERS, COMPONENT_REFERENCES, COMPONENT_RELATIONS, CONNECTIONS,
DOMAINS, EDIT_DISPLAYS, FILES, GRAPHS, GRIDS, IMAGES, INTERFACES, LISTS,
MENUS, METHODS, PROGRAMS, RECORDS, REFERENCE_ARRAYS, REFRESHES, RELATIONS,
RESPONSE_CODES, SOURCES, TEMPLATES, UNITS, VALUE_ARRAYS, VARIABLES,
VARIABLE_LISTS, WAVEFORMS, WRITE_AS_ONES, reference*,
[DELETE identifiant]*,
[REDEFINE identifiant]*,
attribute-redefinition*))
  
```

Les attributs sont spécifiés dans le Tableau 97.

Tableau 97 – Importation de description dun dispositif

Usage	Attribut	Description
o	attribute-redefinition	Voir 7.19.2.
o	AXES	Toutes les instances d'AXIS d'une EDD externe sont importées.
o	BLOCKS_A	Toutes les instances de BLOCK_A d'une EDD externe sont importées.
o	BLOCKS_B	Toutes les instances de BLOCK_B d'une EDD externe sont importées.
o	CHARTS	Toutes les instances de CHART d'une EDD externe sont importées.
o	COLLECTIONS	Toutes les instances de COLLECTION d'une EDD externe sont importées.
o	COMMANDS	Toutes les instances de COMMAND d'une EDD externe sont importées.
o	COMPONENTS	Toutes les instances de COMPONENT d'une EDD externe sont importées.
o	COMPONENT_FOLDERS	Toutes les instances de COMPONENT_FOLDER d'une EDD externe sont importées.
o	COMPONENT_REFERENCES	Toutes les instances de COMPONENT_REFERENCE d'une EDD externe sont importées.
o	COMPONENT_RELATIONS	Toutes les instances de COMPONENT_RELATION d'une EDD externe sont importées.
o	CONNECTIONS	Toutes les instances de CONNECTION d'une EDD externe sont importées.
m	DD_REVISION	Voir 7.2.2.1.
o	DELETE	Une instance importée doit être supprimée.
m	DEVICE_REVISION	Voir 7.2.2.2.
m	DEVICE_TYPE	Voir 7.2.2.3.
o	DOMAINS	Toutes les instances de DOMAIN d'une EDD externe sont importées.
o	EDD_PROFILE	Voir 7.2.2.4.
o	EDD_VERSION	Voir 7.2.2.5.
o	EDIT_DISPLAYS	Toutes les instances d'EDIT_DISPLAY d'une EDD externe sont importées.
o	EVERYTHING	Tous les éléments d'une EDD externe sont importés. Si cette forme est utilisée, aucun type d'élément et aucune instance individuelle ne peut être sélectionnée. Les attributs DELETE, REDEFINE et attribute-redefinition peuvent être utilisés.
o	FILE_NAME	Chaîne qui contient un nom de fichier et son chemin d'accès dans une structure de répertoire. Le fichier contient l'EDD qui doit être importée. En utilisant ce type de référencement, la spécification des informations d'identification d'EDD n'est pas nécessaire.
o	FILES	Toutes les instances de FILE d'une EDD externe sont importées.
o	GRAPHS	Toutes les instances de GRAPH d'une EDD externe sont importées.
o	GRIDS	Toutes les instances de GRID d'une EDD externe sont importées.
o	IMAGES	Toutes les instances d'IMAGE d'une EDD externe sont importées.
o	INTERFACES	Toutes les instances d'INTERFACE d'une EDD externe sont importées.
o	LISTS	Toutes les instances de LIST d'une EDD externe sont importées.
m	MANUFACTURER	Voir 7.2.2.6.
c	MANUFACTURER_EXT	Voir 7.2.2.7.
o	MENUS	Toutes les instances de MENU d'une EDD externe sont importées.
o	METHODS	Toutes les instances de METHOD d'une EDD externe sont importées.
o	PROGRAMS	Toutes les instances de PROGRAM d'une EDD externe sont importées.
o	RECORDS	Toutes les instances de RECORD d'une EDD externe sont importées.
o	REDEFINE	Tous les attributs d'une instance importée doivent être redéfinis.
o	reference	Une instance unique est importée. Une instance unique ne doit pas être importée si elle est déjà incluse dans une importation de type d'élément.
o	REFERENCE_ARRAYS	Toutes les instances de REFERENCE_ARRAY d'une EDD externe sont importées.
o	REFRESHES	Toutes les instances de REFRESH d'une EDD externe sont importées.
o	RELATIONS	Toutes les instances de REFRESH, UNIT et WRITE_AS_ONE d'une EDD externe sont importées.
o	RESPONSE_CODES	Toutes les instances de RESPONSE_CODE d'une EDD externe sont importées.

Usage	Attribut	Description
o	SOURCES	Toutes les instances de SOURCE d'une EDD externe sont importées.
o	TEMPLATES	Toutes les instances de TEMPLATE d'une EDD externe sont importées.
o	UNITS	Toutes les instances d'UNIT d'une EDD externe sont importées.
o	VALUE_ARRAYS	Toutes les instances de VALUE_ARRAY d'une EDD externe sont importées.
o	VARIABLE_LISTS	Toutes les instances de VARIABLE_LIST d'une EDD externe sont importées.
o	VARIABLES	Toutes les instances de VARIABLE d'une EDD externe sont importées.
o	WAVEFORMS	Toutes les instances de WAVEFORM d'une EDD externe sont importées.
o	WRITE_AS_ONES	Toutes les instances de WRITE_AS_ONE d'une EDD externe sont importées.

7.19.2 Attributs spécifiques

7.19.2.1 Redéfinition d'attributs

Objet

Des attributs d'une instance importée peuvent être ajoutés, supprimés ou redéfinis. L'ajout, la suppression et la redéfinition n'affectent pas tous les attributs d'une construction de l'EDD de l'élément de base.

Les attributs de redéfinition sont spécifiés dans le Tableau 98.

Tableau 98 – Attributs de redéfinition

Usage	Attribut	Description
s	ADD	Ajoute l'attribut spécifié suivant. "A" indique que ADD peut être appliqué à l'attribut dans les tables des règles de redéfinition (voir Tableau 99 à Tableau 129).
s	DELETE	Supprime l'attribut spécifié suivant. "D" indique que DELETE peut être appliqué à l'attribut dans les tables des règles de redéfinition (voir Tableau 99 à Tableau 129).
s	REDEFINE	Redéfinit l'attribut spécifié suivant. "R" indique que REDEFINE peut être appliqué à l'attribut dans les tables des règles de redéfinition (voir Tableau 99 à Tableau 129).

Structure lexicale – AXIS

AXIS identifier, [[DELETE, REDEFINE], [CONSTANT_UNIT, HELP, LABEL, MAX_VALUE, MIN_VALUE, SCALING]]+

Les règles de redéfinition d'AXIS sont spécifiées dans le Tableau 98 et le Tableau 99.

Tableau 99 – Règles de redéfinition pour les attributs d'AXIS

A	D	R	Attribut	Description
	•	•	CONSTANT_UNIT	Élément lexical (voir 7.3.2.1).
	•	•	HELP	Élément lexical (voir 7.36.8).
	•	•	LABEL	Élément lexical (voir 7.36.9).
	•	•	MAX_VALUE	Élément lexical (voir 7.3.2.2).
	•	•	MIN_VALUE	Élément lexical (voir 7.3.2.3).
	•	•	SCALING	Élément lexical (voir 7.3.2.4).

Structure lexicale — BLOCK_A

BLOCK identifier, [[ADD, DELETE, REDEFINE], [CHARACTERISTICS, LABEL, PARAMETERS, parameters-list-element, AXIS_ITEMS, CHART_ITEMS, COLLECTION_ITEMS, EDIT_DISPLAY_ITEMS, FILE_ITEMS, GRAPH_ITEMS, GRID_ITEMS, HELP, IMAGE_ITEMS, LIST_ITEMS, LOCAL_PARAMETERS, local-parameters-list-element, MENU_ITEMS, METHOD_ITEMS, PARAMETER_LISTS, parameter-lists-list-element, REFERENCE_ARRAY_ITEMS, REFRESH_ITEMS, SOURCE_ITEMS, UNIT_ITEMS, WAVEFORM_ITEMS, WRITE_AS_ONE_ITEMS, CHARTS, charts-element, LISTS, lists-element, GRAPHS, graphs-element, GRIDS, grids-element, MENUS, menus-element, METHODS, methods-element]]+

Les règles de redéfinition de BLOCK_A sont spécifiées dans le Tableau 98 et le Tableau 100.

Tableau 100 – Règles de redéfinition pour les attributs de BLOCK_A

A	D	R	Attribut	Description
		•	CHARACTERISTICS	Élément lexical (voir 7.4.1.2.1).
		•	LABEL	Élément lexical (voir 7.36.9).
		•	PARAMETERS	Élément lexical (voir 7.4.1.2.2).
•		•	parameters-list-element	La liste d'énumération des éléments individuels de PARAMETERS peut être modifiée.
	•	•	AXIS_ITEMS	Élément lexical (voir 7.4.1.2.3).
	•	•	CHART_ITEMS	Élément lexical (voir 7.4.1.2.4).
	•	•	COLLECTION_ITEMS	Élément lexical (voir 7.4.1.2.5).
	•	•	EDIT_DISPLAY_ITEMS	Élément lexical (voir 7.4.1.2.6).
	•	•	FILE_ITEMS	Élément lexical (voir 7.4.1.2.7).
	•	•	GRAPH_ITEMS	Élément lexical (voir 7.4.1.2.8).
	•	•	GRID_ITEMS	Éléments lexicaux (voir 7.4.1.2.9).
	•	•	HELP	Élément lexical (voir 7.36.8).
	•	•	IMAGE_ITEMS	Éléments lexicaux (voir 7.4.1.2.10).
	•	•	LIST_ITEMS	Élément lexical (voir 7.4.1.2.11).
		•	LOCAL_PARAMETERS	Élément lexical (voir 7.4.1.2.12).
•		•	local-parameters-list-element	La liste d'énumération des éléments individuels de LOCAL_PARAMETERS peut être modifiée.
	•	•	MENU_ITEMS	Élément lexical (voir 7.4.1.2.13).
	•	•	METHOD_ITEMS	Élément lexical (voir 7.4.1.2.14).
		•	PARAMETER_LISTS	Élément lexical (voir 7.4.1.2.15).
•		•	parameter-lists-list-element	La liste d'énumération des éléments individuels de PARAMETERS_LISTS peut être modifiée.
	•	•	REFERENCE_ARRAY_ITEMS	Élément lexical (voir 7.4.1.2.16).
	•	•	REFRESH_ITEMS	Élément lexical (voir 7.4.1.2.17).
	•	•	SOURCE_ITEMS	Élément lexical (voir 7.4.1.2.18).
	•	•	UNIT_ITEMS	Élément lexical (voir 7.4.1.2.19).
	•	•	WAVEFORM_ITEMS	Élément lexical (voir 7.4.1.2.20).
	•	•	WRITE_AS_ONE_ITEMS	Élément lexical (voir 7.4.1.2.21).
	•	•	CHARTS	Élément lexical (voir 7.4.1.2.22).
•	•	•	charts-element	Les éléments individuels de CHARTS peuvent être modifiés.
	•	•	LISTS	Élément lexical (voir 7.4.1.2.23).
•	•	•	lists-element	Les éléments individuels de LISTS peuvent être modifiés.
	•	•	GRAPHS	Élément lexical (voir 7.4.1.2.24).

A	D	R	Attribut	Description
•	•	•	graphs-element	Les éléments individuels de GRAPH peuvent être modifiés.
	•	•	GRIDS	Élément lexical (voir 7.4.1.2.25).
•	•	•	grids-element	Les éléments individuels de GRID peuvent être modifiés.
	•	•	MENUS	Élément lexical (voir 7.4.1.2.26).
•	•	•	menus-element	Les éléments individuels de MENU peuvent être modifiés.
	•	•	METHODS	Élément lexical (voir 7.4.1.2.27).
•	•	•	methods-element	Les éléments individuels de METHOD peuvent être modifiés.

Structure lexicale — BLOCK_B

BLOCK identifier, [REDEFINE, [NUMBER, TYPE]]+

Les règles de redéfinition de BLOCK_B sont spécifiées dans le Tableau 98 et le Tableau 101.

Tableau 101 – Règles de redéfinition pour les attributs de BLOCK_B

A	D	R	Attribut	Description
		•	NUMBER	Élément lexical (voir 7.4.2.2.1).
		•	TYPE	Élément lexical (voir 7.4.2.2.2).

Structure lexicale — CHART

CHART identifier, [[ADD, DELETE, REDEFINE], [MEMBERS, members-list-element, CYCLE_TIME, HEIGHT, HELP, LABEL, LENGTH, TYPE, VALIDITY, WIDTH]]+

Les règles de redéfinition de CHART sont spécifiées dans le Tableau 98 et le Tableau 102.

Tableau 102 – Règles de redéfinition pour les attributs de CHART

A	D	R	Attribut	Description
		•	MEMBERS	Élément lexical (voir 7.36.12).
•	•	•	members-list-element	La liste d'énumération des éléments individuels de MEMBERS peut être modifiée.
	•	•	CYCLE_TIME	Élément lexical (voir 7.5.2.1).
	•	•	HEIGHT	Élément lexical (voir 7.36.7).
	•	•	HELP	Élément lexical (voir 7.36.8).
	•	•	LABEL	Élément lexical (voir 7.36.9).
	•	•	LENGTH	Élément lexical (voir 7.5.2.2).
	•	•	TYPE	Élément lexical (voir 7.5.2.3).
	•	•	VALIDITY	Élément lexical (voir 7.36.16).
	•	•	WIDTH	Élément lexical (voir 7.36.15).

Structure lexicale — COLLECTION

COLLECTION identifier, [[ADD, DELETE, REDEFINE], [MEMBERS, members-list-element, HELP, LABEL, VALIDITY]]+

Les règles de redéfinition de COLLECTION sont spécifiées dans le Tableau 98 et le Tableau 103.

Tableau 103 – Règles de redéfinition pour les attributs de COLLECTION

A	D	R	Attribut	Description
		•	MEMBERS	Élément lexical (voir 7.36.12).
•	•	•	members-list-element	La liste d'énumération des éléments individuels de MEMBERS peut être modifiée.
	•	•	HELP	Élément lexical (voir 7.36.8).
	•	•	LABEL	Élément lexical (voir 7.36.9).
	•	•	VALIDITY	Élément lexical (voir 7.36.16).

Structure lexicale — COMMAND

COMMAND *identifieur*, [[ADD, DELETE, REDEFINE], [OPERATION, TRANSACTION, INDEX, BLOCK_B, NUMBER, SLOT, SUB_SLOT, CONNECTION, HEADER, MODULE, RESPONSE_CODES, response-codes-list-element]]+

Les règles de redéfinition de COMMAND sont spécifiées dans le Tableau 98 et le Tableau 104.

Tableau 104 – Règles de redéfinition pour les attributs de COMMAND

A	D	R	Attribut	Description
			OPERATION	Élément lexical (voir 7.7.2.1).
•	•	•	TRANSACTION	Élément lexical (voir 7.7.2.2).
	•	•	INDEX	Élément lexical (voir 7.7.2.3).
			BLOCK_B	Élément lexical (voir 7.7.2.4).
			NUMBER	Élément lexical (voir 7.7.2.5).
	•	•	SLOT	Élément lexical (voir 7.7.2.6).
	•	•	SUB_SLOT	Élément lexical (voir 7.7.2.7).
	•	•	CONNECTION	Élément lexical (voir 7.7.2.7).
	•	•	HEADER	Élément lexical (voir 7.7.2.9).
	•	•	MODULE	Élément lexical (voir 7.7.2.10).
	•	•	RESPONSE_CODES	Élément lexical (voir 7.36.13).
•	•	•	réponse-codes-list-element	La liste d'énumération des éléments individuels de RESPONSE_CODE peut être modifiée.

Structure lexicale — COMPONENT

COMPONENT *identifieur*, [[DELETE, REDEFINE], [LABEL, CAN_DELETE, CHECK_CONFIGURATION, CLASSIFICATION, HELP, COMPONENT_PARENT, COMPONENT_PATH, COMPONENT_RELATIONS, DECLARATION, DETECT, EDD, INITIAL_VALUES, PROTOCOL, REDUNDANCY, SCAN, SCAN_LIST, SUPPLIED_INTERFACE]]+

Les règles de redéfinition de COMPONENT sont spécifiées dans le Tableau 98 et le Tableau 105.

Tableau 105 – Règles de redéfinition pour les attributs de COMPONENT

A	D	R	Attribut	Description
		•	LABEL	Élément lexical (voir 7.36.9).
	•	•	CAN_DELETE	Élément lexical (voir 7.8.2.1).
	•	•	CHECK_CONFIGURATION	Élément lexical (voir 7.8.2.2).
	•	•	CLASSIFICATION	Élément lexical (voir 7.36.1).

A	D	R	Attribut	Description
	•	•	COMPONENT_PARENT	Élément lexical (voir 7.36.2).
	•	•	COMPONENT_PATH	Élément lexical (voir 7.36.3).
	•	•	COMPONENT_RELATIONS	Élément lexical (voir 7.8.2.3).
	•	•	DECLARATION	Élément lexical (voir 7.8.2.4).
	•	•	DETECT	Élément lexical (voir 7.8.2.5).
	•	•	EDD	Élément lexical (voir 7.8.2.6).
	•	•	HELP	Élément lexical (voir 7.36.8).
	•	•	INITIAL_VALUES	Élément lexical (voir 7.8.2.7).
	•	•	PROTOCOL	Élément lexical (voir 7.36.13).
	•	•	REDUNDANCY	Élément lexical (voir 7.8.2.8).
	•	•	SCAN	Élément lexical (voir 7.8.2.9).
	•	•	SCAN_LIST	Élément lexical (voir 7.8.2.10).
	•	•	SUPPLIED_INTERFACE	Élément lexical (voir 7.36.15).

Structure lexicale — COMPONENT_FOLDER

COMPONENT_FOLDER *identifieur*, [[DELETE, REDEFINE], [LABEL, CLASSIFICATION, COMPONENT_PARENT, COMPONENT_PATH, HELP, PROTOCOL]]+

Les règles de redéfinition de COMPONENT_FOLDER sont spécifiées dans le Tableau 98 et le Tableau 106.

Tableau 106 – Règles de redéfinition pour les attributs de COMPONENT_FOLDER

A	D	R	Attribut	Description
		•	LABEL	Élément lexical (voir 7.36.9).
	•	•	CLASSIFICATION	Élément lexical (voir 7.36.1).
	•	•	COMPONENT_PARENT	Élément lexical (voir 7.36.2).
	•	•	COMPONENT_PATH	Élément lexical (voir 7.36.3).
	•	•	HELP	Élément lexical (voir 7.36.8).
	•	•	PROTOCOL	Élément lexical (voir 7.36.13).

Structure lexicale — COMPONENT_REFERENCE

COMPONENT_REFERENCE *identifieur*, [[DELETE, REDEFINE], [CLASSIFICATION, COMPONENT_PARENT, COMPONENT_PATH, DEVICE_REVISION, DEVICE_TYPE, MANUFACTURER, PROTOCOL]]+

Les règles de redéfinition de COMPONENT_REFERENCE sont spécifiées dans le Tableau 98 et le Tableau 107.

Tableau 107 – Règles de redéfinition pour les attributs de COMPONENT_REFERENCE

A	D	R	Attribut	Description
	•	•	CLASSIFICATION	Élément lexical (voir 7.36.1).
	•	•	COMPONENT_PARENT	Élément lexical (voir 7.36.2).
	•	•	COMPONENT_PATH	Élément lexical (voir 7.36.3).
	•	•	DEVICE_REVISION	Élément lexical (voir 7.2.2.2).
	•	•	DEVICE_TYPE	Élément lexical (voir 7.2.2.3).
	•	•	MANUFACTURER	Élément lexical (voir 7.2.2.6).
	•	•	PROTOCOL	Élément lexical (voir 7.36.13).

Structure lexicale — COMPONENT_RELATION

COMPONENT_RELATION *identifieur*, [[ADD, DELETE, REDEFINE], [COMPONENTS, AUTO_CREATE, FILTER, REQUIRED_RANGES, LABEL, RELATION_TYPE, ADDRESSING, HELP, MAXIMUM_NUMBER, MINIMUM_NUMBER, REQUIRED_INTERFACE, SUPPLIED_INTERFACE]]+

Les règles de redéfinition de COMPONENT_RELATION sont spécifiées dans le Tableau 98 et le Tableau 108.

Tableau 108 – Règles de redéfinition pour les attributs de COMPONENT_RELATION

A	D	R	Attribut	Description
		•	COMPONENTS	Élément lexical (voir 7.11.2.1).
•	•	•	composants-list-element	Les éléments individuels de COMPONENTS peuvent être modifiés.
		•	AUTO_CREATE	Élément lexical (voir 7.11.2.1).
		•	FILTER	Élément lexical (voir 7.11.2.1).
		•	REQUIRED_RANGES	Élément lexical (voir 7.11.2.1).
		•	LABEL	Élément lexical (voir 7.36.9).
		•	RELATION_TYPE	Élément lexical (voir 7.11.2.2).
		•	ADDRESSING	Élément lexical (voir 7.11.2.3).
		•	HELP	Élément lexical (voir 7.36.8).
		•	MAXIMUM_NUMBER	Élément lexical (voir 7.11.2.4).
		•	MINIMUM_NUMBER	Élément lexical (voir 7.11.2.5).
		•	REQUIRED_INTERFACE	Élément lexical (voir 7.11.2.6).
		•	SUPPLIED_INTERFACE	Élément lexical (voir 7.36.15).

Structure lexicale — CONNECTION

CONNECTION *identifieur*, [REDEFINE, [APPINSTANCE]]+

Les règles de redéfinition de CONNECTION sont spécifiées dans le Tableau 98 et le Tableau 109.

Tableau 109 – Règles de redéfinition pour les attributs de CONNECTION

A	D	R	Attribut	Description
		•	APPINSTANCE	Élément lexical (voir 7.12.2).

Structure lexicale — DOMAIN

DOMAIN *identifieur*, [DELETE, REDEFINE], [HANDLING, RESPONSE_CODES]+

Les règles de redéfinition de DOMAIN sont spécifiées dans le Tableau 98 et le Tableau 110.

Tableau 110 – Règles de redéfinition pour les attributs de DOMAIN

A	D	R	Attribut	Description
		•	HANDLING	Élément lexical (voir 7.36.6).
		•	RESPONSE_CODES	Élément lexical (voir 7.36.13).

Structure lexicale — EDIT_DISPLAY

EDIT_DISPLAY *identifieur*, [[DELETE, REDEFINE], [EDIT_ITEMS, LABEL, DISPLAY_ITEMS, POST_EDIT_ACTIONS, PRE_EDIT_ACTIONS]]+

Les règles de redéfinition d'EDIT_DISPLAY sont spécifiées dans le Tableau 98 et le Tableau 111.

Tableau 111 – Règles de redéfinition pour les attributs d'EDIT_DISPLAY

A	D	R	Attribut	Description
		•	EDIT_ITEMS	Élément lexical (voir 7.14.2.1).
		•	LABEL	Élément lexical (voir 7.36.9).
	•	•	DISPLAY_ITEMS	Élément lexical (voir 7.14.2.2).
	•	•	POST_EDIT_ACTIONS	Élément lexical (voir 7.14.2.3).
	•	•	PRE_EDIT_ACTIONS	Élément lexical (voir 7.14.2.4).

Structure lexicale — FILE

FILE identifier, [[ADD, DELETE, REDEFINE], [MEMBERS, members-list-element, HELP, LABEL]]+

Les règles de redéfinition de FILE sont spécifiées dans le Tableau 98 et le Tableau 112.

Tableau 112 – Règles de redéfinition pour les attributs de FILE

A	D	R	Attribut	Description
		•	MEMBERS	Élément lexical (voir 7.36.12).
•	•	•	members-list-element	La liste d'énumération des éléments individuels de MEMBERS peut être modifiée.
	•	•	HELP	Élément lexical (voir 7.36.8).
	•	•	LABEL	Élément lexical (voir 7.36.9).

Structure lexicale — GRAPH

GRAPH identifier, [[ADD, DELETE, REDEFINE], [MEMBERS, members-list-element, CYCLE_TIME, HEIGHT, HELP, LABEL, VALIDITY, WIDTH, X_AXIS]]+

Les règles de redéfinition de GRAPH sont spécifiées dans le Tableau 98 et le Tableau 113.

Tableau 113 – Règles de redéfinition pour les attributs de GRAPH

A	D	R	Attribut	Description
		•	MEMBERS	Élément lexical (voir 7.36.12).
•	•	•	members-list-element	La liste d'énumération des éléments individuels de MEMBERS peut être modifiée.
	•	•	CYCLE_TIME	Élément lexical (voir 7.16.2.1).
	•	•	HEIGHT	Élément lexical (voir 7.36.7).
	•	•	HELP	Élément lexical (voir 7.36.8).
	•	•	LABEL	Élément lexical (voir 7.36.9).
	•	•	VALIDITY	Élément lexical (voir 7.38.16).
	•	•	WIDTH	Élément lexical (voir 7.36.17).
	•	•	X_AXIS	Élément lexical (voir 7.16.2.2).

Structure lexicale — GRID

GRID identifier, [[ADD, DELETE, REDEFINE], VECTORS, HANDLING, HEIGHT, HELP, LABEL, ORIENTATION, VALIDITY, WIDTH]]+

Les règles de redéfinition de GRAPH sont spécifiées dans le Tableau 98 et le Tableau 114.

Tableau 114 – Règles de redéfinition pour les attributs de GRID

A	D	R	Attribut	Description
		•	VECTORS	Élément lexical (voir 7.17.2.1).
	•	•	HANDLING	Élément lexical (voir 7.36.6).
	•	•	HEIGHT	Élément lexical (voir 7.36.7).
	•	•	HELP	Élément lexical (voir 7.36.8).
	•	•	LABEL	Élément lexical (voir 7.36.9).
	•	•	ORIENTATION	Élément lexical (voir 7.17.2.2).
	•	•	VALIDITY	Élément lexical (voir 7.38.16).
	•	•	WIDTH	Élément lexical (voir 7.36.17).

Structure lexicale — IMAGE

IMAGE identifieur, [[DELETE, REDEFINE], PATH, HELP, LABEL, LINK, VALIDITY]]+

Les règles de redéfinition de GRAPH sont spécifiées dans le Tableau 98 et le Tableau 115.

Tableau 115 – Règles de redéfinition pour les attributs d'IMAGE

A	D	R	Attribut	Description
		•	PATH	Élément lexical (voir 7.18.2.1).
	•	•	HELP	Élément lexical (voir 7.36.8).
	•	•	LABEL	Élément lexical (voir 7.36.9).
	•	•	LINK	Élément lexical (voir 7.18.2.2).
	•	•	VALIDITY	Élément lexical (voir 7.38.16).

Structure lexicale — INTERFACE

INTERFACE identifieur, [[DELETE, REDEFINE], [DECLARATION, LABEL, HELP]]+

Les règles de redéfinition d'INTERFACE sont spécifiées dans le Tableau 98 et le Tableau 116.

Tableau 116 – Règles de redéfinition pour les attributs d'INTERFACE

A	D	R	Attribut	Description
		•	DECLARATION	Élément lexical (voir 7.20.2).
	•	•	LABEL	Élément lexical (voir 7.36.9).
	•	•	HELP	Élément lexical (voir 7.36.8).

Structure lexicale — LIST

LIST identifieur, [[DELETE, REDEFINE], [TYPE, CAPACITY, COUNT, HELP, LABEL, VALIDITY]]+

Les règles de redéfinition de LIST sont spécifiées dans le Tableau 98 et le Tableau 117.

Tableau 117 – Règles de redéfinition pour les attributs de LIST

A	D	R	Attribut	Description
		•	TYPE	Élément lexical (voir 7.22.2.1).
	•	•	CAPACITY	Élément lexical (voir 7.22.2.2).
	•	•	COUNT	Élément lexical (voir 7.22.2.3).

A	D	R	Attribut	Description
	•	•	HELP	Élément lexical (voir 7.36.8).
	•	•	LABEL	Élément lexical (voir 7.36.9).
	•	•	VALIDITY	Élément lexical (voir 7.36.16).

Structure lexicale — MENU

MENU *identifieur*, [[ADD, DELETE, REDEFINE], [ITEMS, *items-list-element*, LABEL, ACCESS, HELP, POST_EDIT_ACTIONS, POST_READ_ACTIONS, POST_WRITE_ACTIONS, PRE_EDIT_ACTIONS, PRE_READ_ACTIONS, PRE_WRITE_ACTIONS, PURPOSE, ROLE, STYLE, VALIDITY]]+

Les règles de redéfinition de MENU sont spécifiées dans le Tableau 98 et le Tableau 118.

Tableau 118 – Règles de redéfinition pour les attributs de MENU

A	D	R	Attribut	Description
		•	ITEMS	Élément lexical (voir 7.23.2.1).
		•	LABEL	Élément lexical (voir 7.36.9).
	•	•	ACCESS	Élément lexical (voir 7.23.2.2).
	•	•	HELP	Élément lexical (voir 7.36.8).
	•	•	POST_EDIT_ACTIONS	Élément lexical (voir 7.23.2.3).
	•	•	POST_READ_ACTIONS	Élément lexical (voir 7.23.2.4).
	•	•	POST_WRITE_ACTIONS	Élément lexical (voir 7.23.2.5).
	•	•	PRE_EDIT_ACTIONS	Élément lexical (voir 7.23.2.6).
	•	•	PRE_READ_ACTIONS	Élément lexical (voir 7.23.2.7).
	•	•	PRE_WRITE_ACTIONS	Élément lexical (voir 7.23.2.8).
	•	•	PURPOSE	Élément lexical (voir 7.23.2.9).
	•	•	ROLE	Élément lexical (voir 7.23.2.10).
	•	•	STYLE	Élément lexical (voir 7.23.2.11).
	•	•	VALIDITY	Élément lexical (voir 7.36.16).

Structure lexicale — METHOD

METHOD *identifieur*, [[DELETE, REDEFINE], [DEFINITION, ACCESS, CLASS, LABEL, HELP, TYPE, VALIDITY]]+

Les règles de redéfinition de METHOD sont spécifiées dans le Tableau 98 et le Tableau 94.

Tableau 119 – Règles de redéfinition pour les attributs de METHOD

A	D	R	Attribut	Description
		•	DEFINITION	Élément lexical (voir 7.36.4).
	•	•	ACCESS	Élément lexical (voir 7.24.2.1).
		•	CLASS	Élément lexical (voir 7.24.2.2).
	•	•	LABEL	Élément lexical (voir 7.36.9).
	•	•	HELP	Élément lexical (voir 7.36.8).
			TYPE	Élément lexical (voir 7.24.2.3).
	•	•	VALIDITY	Élément lexical (voir 7.36.16).

Structure lexicale — PROGRAM

PROGRAM *identifier*, [[DELETE, REDEFINE], [ARGUMENTS, *argument-list-element*, RESPONSE_CODES]]+

Les règles de redéfinition de PROGRAM sont spécifiées dans le Tableau 98 et le Tableau 120.

Tableau 120 – Règles de redéfinition pour les attributs de PROGRAM

A	D	R	Attribut	Description
	•	•	ARGUMENTS	Élément lexical (voir 7.25.2).
	•	•	RESPONSE_CODES	Élément lexical (voir 7.36.13).

Structure lexicale — RECORD

RECORD *identifier*, [[ADD, DELETE, REDEFINE], [MEMBERS, *members-list-element*, LABEL, HELP, RESPONSE_CODES, VALIDITY]]+

Les règles de redéfinition de RECORD sont spécifiées dans le Tableau 98 et le Tableau 121.

Tableau 121 – Règles de redéfinition pour les attributs de RECORD

A	D	R	Attribut	Description
		•	MEMBERS	Élément lexical (voir 7.36.10).
•	•	•	members-list-element	La liste d'énumération des éléments individuels de MEMBERS peut être modifiée.
		•	LABEL	Élément lexical (voir 7.36.9).
	•	•	HELP	Élément lexical (voir 7.36.8).
	•	•	RESPONSE_CODES	Élément lexical (voir 7.36.13).
	•	•	VALIDITY	Élément lexical (voir 7.36.16).

Structure lexicale — REFERENCE_ARRAY

REFERENCE_ARRAY *identifier*, [[ADD, DELETE, REDEFINE], [ELEMENTS, *elements-list-element*, HELP, LABEL, VALIDITY]]+

Les règles de redéfinition de REFERENCE_ARRAY sont spécifiées dans le Tableau 98 et le Tableau 122.

Tableau 122 – Règles de redéfinition pour les attributs de REFERENCE_ARRAY

A	D	R	Attribut	Description
		•	ELEMENTS	Élément lexical (voir 7.27.2).
•	•	•	elements-list-element	La liste d'énumération des éléments individuels d'ELEMENTS peut être modifiée.
	•	•	LABEL	Élément lexical (voir 7.36.9).
	•	•	HELP	Élément lexical (voir 7.36.8).
	•	•	VALIDITY	Élément lexical (voir 7.36.16).

Structure lexicale — RESPONSE_CODES

RESPONSE_CODES *identifier*, [[ADD, DELETE, REDEFINE], *response-code-list-element*]+

Les règles de redéfinition de RESPONSE_CODES sont spécifiées dans le Tableau 98 et le Tableau 123.

Tableau 123 – Règles de redéfinition pour les attributs de RESPONSE_CODES

A	D	R	Attribut	Description
•	•	•	response-code-list-element	La liste d'énumération des éléments individuels RESPONSE_CODE peut être modifiée.

Structure lexicale — SOURCE

SOURCE *identifieur*, [[ADD, DELETE, REDEFINE], [MEMBERS, members-list-element, EMPHASIS, EXIT_ACTIONS, HELP, INIT_ACTIONS, LABEL, LINE_COLOR, LINE_TYPE, REFRESH_ACTIONS, VALIDITY, Y_AXIS]]+

Les règles de redéfinition de SOURCE sont spécifiées dans le Tableau 98 et le Tableau 124.

Tableau 124 – Règles de redéfinition pour les attributs de SOURCE

A	D	R	Attribut	Description
		•	MEMBERS	Élément lexical (voir 7.36.12).
•	•	•	members-list-element	La liste d'énumération des éléments individuels de MEMBERS peut être modifiée.
		•	EMPHASIS	Élément lexical (voir 7.36.5).
		•	EXIT_ACTIONS	Élément lexical (voir 7.30.2.1).
		•	HELP	Élément lexical (voir 7.36.8).
		•	INIT_ACTIONS	Élément lexical (voir 7.30.2.2).
		•	LABEL	Élément lexical (voir 7.36.9).
		•	LINE_COLOR	Élément lexical (voir 7.36.10).
		•	LINE_TYPE	Élément lexical (voir 7.36.11).
		•	REFRESH_ACTIONS	Élément lexical (voir 7.30.2.3).
		•	VALIDITY	Élément lexical (voir 7.36.16).
		•	Y_AXIS	Élément lexical (voir 7.30.2.4).

Structure lexicale — TEMPLATE

TEMPLATE *identifieur*, [[DELETE, REDEFINE], [DEFAULT_VALUES, LABEL, HELP]]+

Les règles de redéfinition de TEMPLATE sont spécifiées dans le Tableau 98 et le Tableau 125.

Tableau 125 – Règles de redéfinition pour les attributs de TEMPLATE

A	D	R	Attribut	Description
		•	DEFAULT_VALUES	Élément lexical (voir 7.31.2).
		•	LABEL	Élément lexical (voir 7.36.9).
		•	HELP	Élément lexical (voir 7.36.8).

Structure lexicale — VALUE_ARRAY

VALUE_ARRAY *identifieur*, [[DELETE, REDEFINE], [LABEL, NUMBER_OF_ELEMENTS, TYPE, HELP, RESPONSE_CODES, VALIDITY]]+

Les règles de redéfinition de VALUE_ARRAY sont spécifiées dans le Tableau 98 et le Tableau 126.

Tableau 126 – Règles de redéfinition pour les attributs de VALUE_ARRAY

A	D	R	Attribut	Description
		•	LABEL	Élément lexical (voir 7.36.9).
		•	NUMBER_OF_ELEMENTS	Élément lexical (voir 7.32.2.1).
		•	TYPE	Élément lexical (voir 7.32.2.2).
	•	•	HELP	Élément lexical (voir 7.36.8).
	•	•	RESPONSE_CODES	Élément lexical (voir 7.36.13).
	•	•	VALIDITY	Élément lexical (voir 7.36.16).

Structure lexicale — VARIABLE

VARIABLE identifier, [[ADD, DELETE, REDEFINE], [CLASS, LABEL, TYPE, DEFAULT_VALUE, DISPLAY_FORMAT, EDIT_FORMAT, INITIAL_VALUE, MAX_VALUE, MIN_VALUE, SCALING_FACTOR, TIME_FORMAT, TIME_SCALE, type-list-element, CONSTANT_UNIT, HANDLING, HELP, POST_EDIT_ACTIONS, POST_READ_ACTIONS, POST_WRITE_ACTIONS, PRE_EDIT_ACTIONS, PRE_READ_ACTIONS, PRE_WRITE_ACTIONS, READ_TIMEOUT, REFRESH_ACTIONS, RESPONSE_CODES, STYLE, VALIDITY, WRITE_TIMEOUT]]+

Les règles de redéfinition sont spécifiées dans le Tableau 98 et le Tableau 127.

Tableau 127 – Règles de redéfinition pour les attributs de VARIABLE

A	D	R	Attribut	Description
		•	CLASS	Élément lexical (voir 7.33.2.1).
	•	•	LABEL	Élément lexical (voir 7.36.9).
			TYPE	Élément lexical (voir 7.33.2.2).
	•	•	DEFAULT_VALUE	Élément lexical (voir 7.33.2.2).
	•	•	DISPLAY_FORMAT	Élément lexical (voir 7.33.2.2).
	•	•	EDIT_FORMAT	Élément lexical (voir 7.33.2.2).
	•	•	INITIAL_VALUE	Élément lexical (voir 7.33.2.2).
	•	•	MAX_VALUE	Élément lexical (voir 7.33.2.2).
	•	•	MIN_VALUE	Élément lexical (voir 7.33.2.2).
	•	•	SCALING_FACTOR	Élément lexical (voir 7.33.2.2).
	•	•	TIME_FORMAT	Élément lexical (voir 7.33.2.2).
	•	•	TIME_SCALE	Élément lexical (voir 7.33.2.2).
•	•	•	type-list-element	La liste d'énumération des éléments individuels de chaque TYPE peut être modifiée.
	•	•	CONSTANT_UNIT	Élément lexical (voir 7.33.2.3).
	•	•	HANDLING	Élément lexical (voir 7.36.6).
	•	•	HELP	Élément lexical (voir 7.36.8).
	•	•	POST_EDIT_ACTIONS	Élément lexical (voir 7.33.2.6).
	•	•	POST_READ_ACTIONS	Élément lexical (voir 7.33.2.7).
	•	•	POST_WRITE_ACTIONS	Élément lexical (voir 7.33.2.8).
	•	•	PRE_EDIT_ACTIONS	Élément lexical (voir 7.33.2.9).
	•	•	PRE_READ_ACTIONS	Élément lexical (voir 7.33.2.10).
	•	•	PRE_WRITE_ACTIONS	Élément lexical (voir 7.33.2.11).
	•	•	READ_TIMEOUT	Élément lexical (voir 7.33.2.12).
	•	•	REFRESH_ACTIONS	Élément lexical (voir 7.33.2.13).

A	D	R	Attribut	Description
	•	•	RESPONSE_CODES	Élément lexical (voir 7.36.13).
	•	•	STYLE	Élément lexical (voir 7.33.2.13).
	•	•	VALIDITY	Élément lexical (voir 7.36.16).
	•	•	WRITE_TIMEOUT	Élément lexical (voir 7.33.2.12).

Structure lexicale — VARIABLE_LIST

VARIABLE_LIST *identifier*, [[ADD, DELETE, REDEFINE], [MEMBERS, members-list-element, HELP, LABEL, RESPONSE_CODES]]+

Les règles de redéfinition de VARIABLE_LIST sont spécifiées dans le Tableau 98 et le Tableau 128.

Tableau 128 – Règles de redéfinition pour les attributs de VARIABLE_LIST

A	D	R	Attribut	Description
		•	MEMBERS	Élément lexical (voir 7.36.12).
•	•	•	members-list-element	La liste d'énumération des éléments individuels de MEMBERS peut être modifiée.
	•	•	HELP	Élément lexical (voir 7.36.8).
	•	•	LABEL	Élément lexical (voir 7.36.9).
	•	•	RESPONSE_CODES	Élément lexical (voir 7.36.13).

Structure lexicale — WAVEFORM

WAVEFORM *identifier*, [[ADD, DELETE, REDEFINE], [TYPE, X_VALUES, Y_VALUES, NUMBER_OF_POINTS, X_INITIAL, X_INCREMENT, EMPHASIS, HANDLING, HELP, INIT_ACTIONS, KEY_POINTS, LABEL, LINE_COLOR, LINE_TYPE, REFRESH_ACTIONS, VALIDITY, Y_AXIS]]+

Les règles de redéfinition de WAVEFORM sont spécifiées dans le Tableau 98 et le Tableau 129.

Tableau 129 – Règles de redéfinition pour les attributs de WAVEFORM

A	D	R	Attribut	Description
		•	TYPE	Élément lexical (voir 7.35.2.1).
•		•	X_VALUES	Élément lexical (voir 7.35.2.1).
•		•	Y_VALUES	Élément lexical (voir 7.35.2.1).
	•	•	NUMBER_OF_POINTS	Élément lexical (voir 7.35.2.1).
		•	X_INITIAL	Élément lexical (voir 7.35.2.1).
		•	X_INCREMENT	Élément lexical (voir 7.35.2.1).
	•	•	EMPHASIS	Élément lexical (voir 7.36.5).
	•	•	EXIT_ACTIONS	Élément lexical (voir 7.35.2.2).
	•	•	HANDLING	Élément lexical (voir 7.36.6).
	•	•	HELP	Élément lexical (voir 7.36.8).
	•	•	INIT_ACTIONS	Élément lexical (voir 7.35.2.3).
	•	•	KEY_POINTS	Élément lexical (voir 7.35.2.4).
	•	•	LABEL	Élément lexical (voir 7.36.9).
	•	•	LINE_COLOR	Élément lexical (voir 7.36.10).
	•	•	LINE_TYPE	Élément lexical (voir 7.36.11).
	•	•	REFRESH_ACTIONS	Élément lexical (voir 7.35.2.5).

A	D	R	Attribut	Description
	•	•	VALIDITY	Élément lexical (voir 7.36.16).
	•	•	Y_AXIS	Élément lexical (voir 7.35.2.6).

7.20 INTERFACE

7.20.1 Structure générale

Objet

L'INTERFACE spécifie un ensemble de constructions de base de composants qui sont accessibles par d'autres composants. Une interface peut contenir des déclarations de constructions de base, par exemple, des VARIABLE, des MENU.

Il convient que toutes les INTERFACE fournies d'un composant soient répertoriées dans l'attribut SUPPLIED_INTERFACE du COMPONENT.

Si des éléments INTERFACE sont définis pour un COMPONENT, l'accès est restreint. D'autres composants peuvent accéder uniquement aux constructions de base de l'EDD (par exemple, les VARIABLE) définies dans une interface.

Si aucune interface n'est définie pour le composant, l'accès est autorisé à toutes les constructions de base.

Structure lexicale

INTERFACE identifieur [LABEL, HELP, DECLARATION]

Les attributs d'INTERFACE sont spécifiés dans le Tableau 130.

Tableau 130 – Attributs d'INTERFACE

Usage	Attribut	Description
m	DECLARATION	Élément lexical (voir 7.20.2).
m	LABEL	Élément lexical (voir 7.36.9).
o	HELP	Élément lexical (voir 7.36.8).

7.20.2 Attribut spécifique – DECLARATION

Objet

L'attribut DECLARATION (déclaration) spécifie les constructions de base incluses dans l'interface.

Structure lexicale

DECLARATION [IMAGE, MENU, METHOD, VARIABLE] +

Les attributs de DECLARATION sont spécifiés dans le Tableau 131.

Tableau 131 – Attributs de DECLARATION

Usage	Attribut	Description
o	IMAGE	Déclarations d'IMAGE (voir 7.18).
o	MENU	Déclarations de MENU (voir 7.23).
o	METHOD	Déclarations de METHOD (voir 7.24).
o	VARIABLE	Déclarations de VARIABLE (voir 7.33).

7.21 LIKE

Objet

LIKE (similaire) est utilisé pour créer une nouvelle instance d'EDD sur la base d'un élément existant. LIKE fait une copie du type d'élément existant avec tous les attributs.

Structure lexicale

LIKE *identifieur*, *reference*, *attribute-redefinition*+, *item-type*

Les attributs sont spécifiés dans le Tableau 132.

Tableau 132 – Attributs de LIKE

Usage	Attribut	Description
m	identifieur	Nom de la nouvelle instance d'EDD.
m	reference	Référence à une instance d'EDD qui doit être copiée.
o	attribute-redefinition	Spécifie si un attribut est ajouté, supprimé ou redéfini (voir 7.19.2). Seuls les attributs existants peuvent être supprimés ou redéfinis.
o	item-type	Un des éléments énumérés dans le Tableau 47. Même si item-type est présent, il n'est pas interprété.

7.22 LIST

7.22.1 Structure générale

Objet

LIST (liste) décrit une séquence d'éléments dans laquelle chaque élément de la séquence est défini par une instance de construction EDDL.

Structure lexicale

LIST *identifieur* [TYPE, CAPACITY, COUNT, HELP, LABEL, VALIDITY]

Les attributs de LIST sont spécifiés dans le Tableau 133.

Tableau 133 – Attributs de LIST

Usage	Attribut	Description
m	TYPE	Élément lexical (voir 7.22.2.1).
o	CAPACITY	Élément lexical (voir 7.22.2.2).
o	COUNT	Élément lexical (voir 7.22.2.3).
o	HELP	Élément lexical (voir 7.36.8).
o	LABEL	Élément lexical (voir 7.36.9).
o	VALIDITY	Élément lexical (voir 7.36.16).

7.22.2 Attributs spécifiques

7.22.2.1 TYPE

Objet

L'attribut TYPE est une référence à une instance d'une construction EDDL (par exemple, VARIABLE, COLLECTION). Tous les attributs de la construction EDDL s'appliquent à chacun des éléments de LIST.

Structure lexicale

TYPE reference

L'attribut de TYPE est spécifié dans le Tableau 134.

Tableau 134 – Attribut de TYPE

Usage	Attribut	Description
m	reference	Référence à une instance de VARIABLE, une instance de RECORD, une instance de VALUE_ARRAY, une instance de COLLECTION ou une instance de REFERENCE_ARRAY.

7.22.2.2 CAPACITY**Objet**

L'attribut CAPACITY (capacité) spécifie le nombre maximal d'éléments qui peuvent être stockés dans la liste. Lorsqu'une liste fait partie d'un FILE (fichier), la CAPACITY ne doit pas être spécifiée étant donné que la capacité est limitée uniquement par les ressources de l'application EDD.

Structure lexicale

CAPACITY unsigned_integer

L'attribut de CAPACITY est spécifié dans le Tableau 135.

Tableau 135 – Attribut de CAPACITY, COUNT

Usage	Attribut	Description
m	unsigned_integer	Constante entière non signée

7.22.2.3 COUNT**Objet**

L'attribut COUNT (compteur) spécifie les nombres d'éléments actuellement stockés dans la liste. Lorsqu'une liste fait partie d'un FILE (fichier), le COUNT ne doit pas être spécifié étant donné que seule l'application EDD connaît le nombre d'éléments actuellement présents dans la liste.

Si COUNT est spécifié, il définit la taille de l'instance de LIST lors de sa création. S'il n'est pas défini, une instance de LIST avec zéro élément est créée.

Structure lexicale

COUNT (unsigned_integer) <exp>

L'attribut de COUNT est spécifié dans le Tableau 135.

7.23 MENU**7.23.1 Structure générale****Objet**

MENU organise dans une structure hiérarchique des variables, des méthodes et d'autres éléments spécifiés dans le langage EDDL. Une application utilisateur peut utiliser les ITEMS (options) de MENU pour afficher et entrer des informations d'une façon organisée et cohérente.

Structure lexicale

MENU identifier [ITEMS, LABEL, ACCESS, HELP, POST_EDIT_ACTIONS, POST_READ_ACTIONS, POST_WRITE_ACTIONS, PRE_EDIT_ACTIONS, PRE_READ_ACTIONS, PRE_WRITE_ACTIONS, PURPOSE, ROLE, STYLE, VALIDITY]

Les attributs de MENU sont spécifiés dans le Tableau 136.

Tableau 136 – Attributs de MENU

Usage	Attribut	Description
m	ITEMS	Élément lexical (voir 7.23.2.1).
m	LABEL	Élément lexical (voir 7.36.9).
o	ACCESS	Élément lexical (voir 7.23.2.2).
o	HELP	Élément lexical (voir 7.36.8).
o	POST_EDIT_ACTIONS	Élément lexical (voir 7.23.2.3).
o	POST_READ_ACTIONS	Élément lexical (voir 7.23.2.4).
o	POST_WRITE_ACTIONS	Élément lexical (voir 7.23.2.5).
o	PRE_EDIT_ACTIONS	Élément lexical (voir 7.23.2.6).
o	PRE_READ_ACTIONS	Élément lexical (voir 7.23.2.7).
o	PRE_WRITE_ACTIONS	Élément lexical (voir 7.23.2.8).
c	PURPOSE	Élément lexical (voir 7.23.2.9).
o	ROLE	Élément lexical (voir 7.23.2.10).
o	STYLE	Élément lexical (voir 7.23.2.11).
o	VALIDITY	Élément lexical (voir 7.36.16).

7.23.2 Attributs spécifiques

7.23.2.1 ITEMS

Objet

L'attribut ITEMS (éléments) spécifie les éléments sélectionnés de l'EDD qui doivent être affichés à l'utilisateur ainsi que des qualificatifs optionnels pour chaque élément. Ces qualificatifs (DISPLAY_VALUE, HIDDEN, READ_ONLY, NO_LABEL, NO_UNIT, REVIEW) sont utilisés pour contrôler le traitement des éléments.

NOTE 1 Les éléments de MENU sont présentés à l'utilisateur dans l'ordre dans lequel ils apparaissent.

NOTE 2 Pour des menus verticaux, le premier élément est affiché en haut et le dernier élément en bas. Pour des menus horizontaux, le premier élément est affiché sur la gauche et le dernier élément sur la droite.

COLUMNBREAK (saut de colonne) et ROWBREAK (saut de ligne) ne doivent pas apparaître dans une instruction conditionnelle.

Structure lexicale

ITEMS [(reference, DISPLAY_VALUE, HIDDEN, READ_ONLY, NO_LABEL, NO_UNIT, REVIEW)<exp>, SEPARATOR, ROWBREAK, COLUMNBREAK, string]+

Les attributs d'ITEMS sont spécifiés dans le Tableau 137.

Tableau 137 – Attributs d'ITEMS

Usage	Attribut	Description
m	reference	Référence à une VARIABLE, un BLOCK_A PARAMETERS, un élément de BLOCK_A PARAMETERS, un CHART, une COLLECTION, un EDIT_DISPLAY, un GRAPH, une GRID, une IMAGE, une LIST, un MENU, une METHOD, une METHOD avec arguments, un RECORD, un REFERENCE_ARRAY, ou un VALUE_ARRAY.
c/o	DISPLAY_VALUE	Utilisé uniquement avec VARIABLE; « o » utilisé dans tous les autres cas. Spécifie que la valeur de la VARIABLE correspondante est affichée en plus de son LABEL.
o	HIDDEN	L'élément n'est pas affiché.
c/o	READ_ONLY	Utilisé uniquement avec VARIABLE; « o » utilisé dans tous les autres cas. Spécifie que la valeur doit être affichée mais qu'elle ne peut pas être modifiée (indépendamment de son attribut HANDLING).
c/o	NO_LABEL	Utilisé uniquement avec VARIABLE; « o » utilisé dans tous les autres cas. Spécifie que l'étiquette de la variable n'est pas affichée; DISPLAY_VALUE est implicite.
c/o	NO_UNIT	Utilisé uniquement avec VARIABLE; « o » utilisé dans tous les autres cas. Spécifie que l'unité technique de la variable n'est pas affichée; DISPLAY_VALUE est implicite.
c/o	REVIEW	Utilisé uniquement avec MENU; « o » utilisé dans tous les autres cas. REVIEW est utilisé pour présenter une grande quantité de données, par exemple, pour les passer en revue.
c/o	INLINE	Utilisé uniquement avec IMAGE; « o » utilisé dans tous les autres cas. Si l'IMAGE est dans un MENU avec un attribut STYLE égal à WINDOW, DIALOG, PAGE ou GROUP et le qualificatif INLINE n'est pas spécifié, il convient que l'IMAGE couvre l'étendue du MENU; sinon, il convient que l'IMAGE soit affichée en ligne avec les autres ITEMS du MENU.
o	SEPARATOR	Il convient que les éléments de menu avant et après le SEPARATOR soient séparés, par exemple, par une ligne horizontale.
c/o	COLUMNBREAK	Utilisé uniquement dans un MENU dont l'attribut STYLE est WINDOW, DIALOG, PAGE, ou GROUP; spécifie qu'il convient qu'une nouvelle colonne commence.
c/o	ROWBREAK	Utilisé uniquement dans un MENU dont l'attribut STYLE est WINDOW, DIALOG, PAGE, ou GROUP; spécifie qu'il convient qu'une nouvelle ligne commence.
o	string	Texte statique qu'il convient d'afficher sur le menu.

7.23.2.2 ACCESS**Objet**

L'attribut ACCESS spécifie si les éléments affichés ou utilisés dans le MENU sont lus depuis le dispositif (ONLINE) ou localement (OFFLINE). Si aucun attribut ACCESS n'est défini, ONLINE est la valeur par défaut.

Structure lexicale

ACCESS [(OFFLINE, ONLINE) <exp>]

Les attributs d'ACCESS sont spécifiés dans le Tableau 138.

Tableau 138 – Attributs d'ACCESS

Usage	Attribut	Description
s	OFFLINE	Les valeurs des éléments affichés dans le MENU sont lues localement.
s	ONLINE	Il convient que les valeurs des éléments affichés dans le MENU soient issues du dispositif et réactualisées à un intervalle approprié.

NOTE L'implémentation est responsable de décider à quel moment il convient d'effectuer le transfert de valeurs modifiées vers un dispositif (typiquement après l'exécution d'une entrée unique ou un MENU pour un attribut ONLINE ACCESS, ou via un menu séparé pour mettre à jour le dispositif).

7.23.2.3 POST_EDIT_ACTIONS

Objet

L'attribut POST_EDIT_ACTIONS (actions post-édition) spécifie les METHOD qui doivent être exécutées après que l'utilisateur ait fini le traitement du MENU. Si le MENU est abandonné, les POST_EDIT_ACTIONS ne sont pas exécutées. Les METHOD spécifiées doivent être exécutées dans l'ordre dans lequel elles apparaissent. Si une METHOD se termine pour une raison imprévue, les METHOD suivantes ne sont pas exécutées.

Les POST_EDIT_ACTIONS éventuellement associées à une entité référencée dans les ITEMS doivent être exécutées lorsqu'une nouvelle valeur est entrée pour l'entité. Les POST_EDIT_ACTIONS du MENU sont exécutées uniquement après que toutes les POST_EDIT_ACTIONS spécifiques à l'entité soient terminées.

L'attribut POST_EDIT_ACTIONS ne doit pas être exprimé en utilisant des instructions conditionnelles IF, IF-ELSE, SELECT. Si un comportement conditionnel est requis, les méthodes appelées par l'action doivent être compatibles avec celui-ci.

Structure lexicale

POST_EDIT_ACTIONS [reference]+ DEFINITION

Les attributs de POST_EDIT_ACTIONS sont spécifiés dans le Tableau 139.

Tableau 139 – Attributs POST_EDIT_ACTIONS, PRE_EDIT_ACTIONS, POST_READ_ACTIONS, PRE_READ_ACTIONS, POST_WRITE_ACTIONS, PRE_WRITE_ACTIONS

Usage	Attribut	Description
o	reference	Référence à une instance de METHOD ou une instance de METHOD avec des arguments.
o	DEFINITION	Élément lexical (voir 7.36.4).

7.23.2.4 POST_READ_ACTIONS

Objet

L'attribut POST_READ_ACTIONS (actions post-lecture) spécifie les METHOD qui doivent être exécutées après que toutes les entités référencées dans l'attribut ITEMS aient été lues et traitées. Les METHOD spécifiées doivent être exécutées dans l'ordre dans lequel elles apparaissent. Si une METHOD se termine pour une raison imprévue, les METHOD suivantes ne sont pas exécutées et l'activation de MENU est annulée.

Les POST_READ_ACTIONS éventuellement associées à une entité référencée dans les ITEMS doivent être exécutées lorsqu'une nouvelle valeur est lue pour l'entité. Les POST_READ_ACTIONS du MENU doivent être exécutées uniquement après que toutes les POST_READ_ACTIONS spécifiques à l'entité soient terminées.

L'attribut POST_READ_ACTIONS ne doit pas être exprimé en utilisant des instructions conditionnelles (IF, IF-ELSE, SELECT). Si un comportement conditionnel est requis, les méthodes appelées par l'action doivent être compatibles avec celui-ci.

Structure lexicale

POST_READ_ACTIONS [reference]+, DEFINITION

Les attributs de POST_READ_ACTIONS sont spécifiés dans le Tableau 139.

7.23.2.5 POST_WRITE_ACTIONS**Objet**

L'attribut POST_WRITE_ACTIONS (actions post-écriture) spécifie les METHOD qui doivent être exécutées après que toutes les entités référencées dans l'attribut ITEMS aient été écrites et traitées. Les METHOD spécifiées doivent être exécutées dans l'ordre dans lequel elles apparaissent. Si une METHOD se termine pour une raison imprévue, les METHOD suivantes ne sont pas exécutées.

Les POST_WRITE_ACTIONS éventuellement associées à une entité référencée dans les ITEMS doivent être exécutées lorsqu'une nouvelle valeur est écrite pour l'entité. Les POST_WRITE_ACTIONS du MENU doivent être exécutées uniquement après que toutes les POST_WRITE_ACTIONS spécifiques à l'entité soient terminées.

L'attribut POST_WRITE_ACTIONS ne doit pas être exprimé en utilisant des instructions conditionnelles (IF, IF-ELSE, SELECT). Si un comportement conditionnel est requis, les méthodes appelées par l'action doivent être compatibles avec celui-ci.

Structure lexicale

POST_WRITE_ACTIONS [reference]+, DEFINITION

Les attributs de POST_WRITE_ACTIONS sont spécifiés dans le Tableau 139.

7.23.2.6 PRE_EDIT_ACTIONS**Objet**

L'attribut PRE_EDIT_ACTIONS (actions pré-édition) spécifie les METHOD qui doivent être exécutées immédiatement après l'activation du MENU. Les METHOD spécifiées doivent être exécutées dans l'ordre dans lequel elles apparaissent. Si une METHOD se termine pour une raison imprévue, les METHOD suivantes ne sont pas exécutées et l'activation du MENU est annulée.

Les PRE_EDIT_ACTIONS du MENU doivent être exécutées avant l'une quelconque des PRE_EDIT_ACTIONS définies des entités référencées. Une fois que les PRE_EDIT_ACTIONS de MENU sont terminées, les PRE_EDIT_ACTIONS éventuelles pour les entités référencées doivent être exécutées.

L'attribut PRE_EDIT_ACTIONS ne doit pas être exprimé en utilisant une instruction conditionnelle (IF, IF-ELSE, SELECT). Si un comportement conditionnel est requis, les méthodes appelées par l'action doivent être compatibles avec celui-ci.

Structure lexicale

PRE_EDIT_ACTIONS [reference]+, DEFINITION

Les attributs de PRE_EDIT_ACTIONS sont spécifiés dans le Tableau 139.

7.23.2.7 PRE_READ_ACTIONS**Objet**

L'attribut PRE_READ_ACTIONS (actions pré-lecture) spécifie les METHOD qui doivent être exécutées avant que toutes les entités référencées dans l'attribut ITEMS soient lues ou traitées. Les METHOD spécifiées doivent être exécutées dans l'ordre dans lequel elles

apparaissent. Si une METHOD se termine pour une raison imprévue, les METHOD suivantes ne sont pas exécutées et l'activation de MENU est annulée. Une fois que les PRE_READ_ACTIONS de MENU sont terminées, les PRE_READ_ACTIONS éventuelles pour les entités référencées doivent être exécutées.

L'attribut PRE_READ_ACTIONS ne doit pas être exprimé en utilisant des instructions conditionnelles (IF, IF-ELSE, SELECT). Si un comportement conditionnel est requis, les méthodes appelées par l'action doivent être compatibles avec celui-ci.

Structure lexicale

PRE_READ_ACTIONS [reference]+, DEFINITION

Les attributs de PRE_READ_ACTIONS sont spécifiés dans le Tableau 139.

7.23.2.8 PRE_WRITE_ACTIONS

Objet

L'attribut PRE_WRITE_ACTIONS (actions pré-écriture) spécifie les METHOD qui doivent être exécutées avant que toutes les entités référencées dans l'attribut ITEMS soient écrites ou traitées. Les METHOD spécifiées doivent être exécutées dans l'ordre dans lequel elles apparaissent. Si une METHOD se termine pour une raison imprévue, les METHOD suivantes ne sont pas exécutées. Une fois que les PRE_WRITE_ACTIONS de MENU sont terminées, les PRE_WRITE_ACTIONS pour les entités référencées doivent être exécutées.

L'attribut PRE_WRITE_ACTIONS ne doit pas être exprimé en utilisant une instruction conditionnelle (IF, IF-ELSE, SELECT). Si un comportement conditionnel est requis, les méthodes appelées par l'action doivent être compatibles avec celui-ci.

Structure lexicale

PRE_WRITE_ACTIONS [reference]+, DEFINITION

Les attributs de PRE_WRITE_ACTIONS sont spécifiés dans le Tableau 139.

7.23.2.9 PURPOSE

Objet

L'attribut PURPOSE (objet) définit l'objet d'application du MENU. Celui-ci indique comment le MENU est utilisé, par exemple, pour une présentation tabulaire, des actions de diagnostic et une liste de transfert/téléchargement.

Structure lexicale

PURPOSE [(CATALOG, DIAGNOSE, LOAD_TO_APPLICATION, LOAD_TO_DEVICE, MENU, PROCESS_VALUE, SIMULATION, string, TABLE)<exp>]+

Les attributs de PURPOSE sont spécifiés dans le Tableau 140.

Tableau 140 – Attributs de PURPOSE

Usage	Attribut	Description
s	CATALOG	Les éléments de l'attribut ITEMS d'une instance de MENU sont des références à des informations spécifiques au catalogue (par exemple, une variable d'EDD contenant le numéro d'ordre).
s	DIAGNOSE	Le MENU est utilisé pour des actions de diagnostic.
s	LOAD_TO_APPLICATION	Le MENU est utilisé pour répertorier les VARIABLE dont les valeurs sont envoyées du dispositif vers une application.
s	LOAD_TO_DEVICE	Le MENU est utilisé pour répertorier des VARIABLE dont les valeurs sont envoyées au dispositif.

Usage	Attribut	Description
s	MENU	MENU qui contient d'autres options de MENU ou éléments de contrôle normalement présentés en utilisant des lignes horizontales.
s	PROCESS_VALUE	Le MENU est utilisé pour afficher des valeurs de processus.
s	SIMULATION	Le MENU est utilisé pour des actions de simulation.
s	string	PURPOSE défini par l'utilisateur pour le MENU.
s	TABLE	MENU qui contient d'autres éléments ou VARIABLE de MENU normalement présentés en utilisant des colonnes verticales.

NOTE Il convient que l'attribut PURPOSE de MENU soit utilisé pour une représentation de données et de fonction horizontale (par exemple, des touches de fonction, des menus d'application). Il convient qu'un élément TABLE soit utilisé pour une représentation des données d'une fonction verticale (par exemple, des tableaux).

7.23.2.10 ROLE

Objet

L'attribut ROLE (rôle) spécifie dans quel contexte de rôle le MENU est utilisé. Si un MENU contient des sous-menus, seuls ces sous-menus qui ont le même ROLE sont affichés. Si un MENU n'a pas d'attribut ROLE, le MENU doit être affiché lorsqu'il est référencé.

Structure lexicale

ROLE [(string)<exp>]+

L'attribut de ROLE est spécifié dans le Tableau 141.

Tableau 141 – Attribut de ROLE

Usage	Attribut	Description
m	string	Chaîne qui indique le ROLE du MENU.

NOTE Une EDD peut contenir un ou plusieurs ROLE suivants: MAINTENANCE (maintenance), SPECIALIST (spécialiste), DIAGNOSE (diagnostic), HANDHELD (portatif), SERVICE (service), MANUFACTURER (fabricant), OBSERVER (observateur).

7.23.2.11 STYLE

Objet

L'attribut STYLE spécifie la manière dont le MENU est affiché.

Lorsque l'attribut STYLE est DIALOG (boîte de dialogue), WINDOW (fenêtre), PAGE ou GROUP (groupe), le qualificatif DISPLAY_VALUE (voir 7.23.2.1) est appliqué à chaque élément sur le MENU.

Si un ou plusieurs MENU sont actifs et un MENU DIALOG (boîte de dialogue) est activé, le MENU DIALOG est prioritaire et doit être fermé avant que tout autre MENU puisse être utilisé.

Structure lexicale

STYLE (DIALOG, WINDOW, PAGE, GROUP, MENU, TABLE, string)

Les attributs de STYLE sont spécifiés dans le Tableau 142.

Tableau 142 – Attributs de STYLE

Usage	Attribut	Description
s	DIALOG	Aucun autre MENU ne peut être activé lorsque ce MENU, ou l'un quelconque de ses sous-menus, est actif. Le MENU doit être fermé avant que des interactions avec d'autres MENU soient possibles, c'est-à-dire, il est spécifié qu'il convient que les éléments du MENU soient affichés sous forme de boîte de dialogue modale.
s	WINDOW	D'autres MENU peuvent être activés pendant que ce MENU est actif. Les interactions avec tous les MENU activés ayant des attributs de STYLE WINDOW (fenêtre) sont possibles, c'est-à-dire, il est spécifié qu'il convient que les éléments du MENU soient affichés en tant que fenêtre non modale.
s	PAGE	Il est recommandé que les éléments du MENU soient affichés sous forme de page de propriétés à onglets.
s	GROUP	Il est recommandé que les éléments du MENU soient affichés sous forme de cadre de groupe.
s	MENU	Il est recommandé que les éléments du MENU soient affichés sous forme de menu déroulant ou contextuel.
s	TABLEAU	Il est recommandé que les éléments du MENU soient affichés en colonnes verticales.
s	string	Style de MENU spécifique à un outil. Si un outil ne peut pas interpréter la chaîne, cet attribut STYLE doit être ignoré.

7.23.3 Schémas de séquence pour les actions

La Figure 28 présente des schémas de séquence pour des actions associées aux MENU et leurs VARIABLE et METHOD contenues. Les actions sont exécutées et terminées en séquence du haut vers le bas du schéma. Si une action échoue ou est abandonnée pour une raison imprévue, les actions restantes ne sont pas exécutées et le MENU est annulé.

Lorsque le MENU a un ACCESS OFFLINE (hors connexion, données provenant d'une base de données locale), les actions pré/post-lecture/écriture pour la VARIABLE ne sont pas exécutées.

Lorsque le MENU a un ACCESS ONLINE (en ligne, données lues depuis ou écrites vers le dispositif), les actions pré/post-lecture/écriture pour la VARIABLE doivent être exécutées dans l'ordre indiqué dans la séquence.

Les actions pré/post-édition doivent toujours être exécutées en cas d'ACCESS OFFLINE et ONLINE.

Dans le cas de l'accès en ligne, les valeurs de VARIABLE peuvent être modifiées et immédiatement écrites sauf si elles sont impliquées dans une relation WRITE_AS_ONE. Les entités WRITE_AS_ONE éventuelles sont écrites ensemble. Une fois que les VARIABLES WRITE_AS_ONE ont été écrites, les POST_WRITE_ACTIONS sont exécutées.

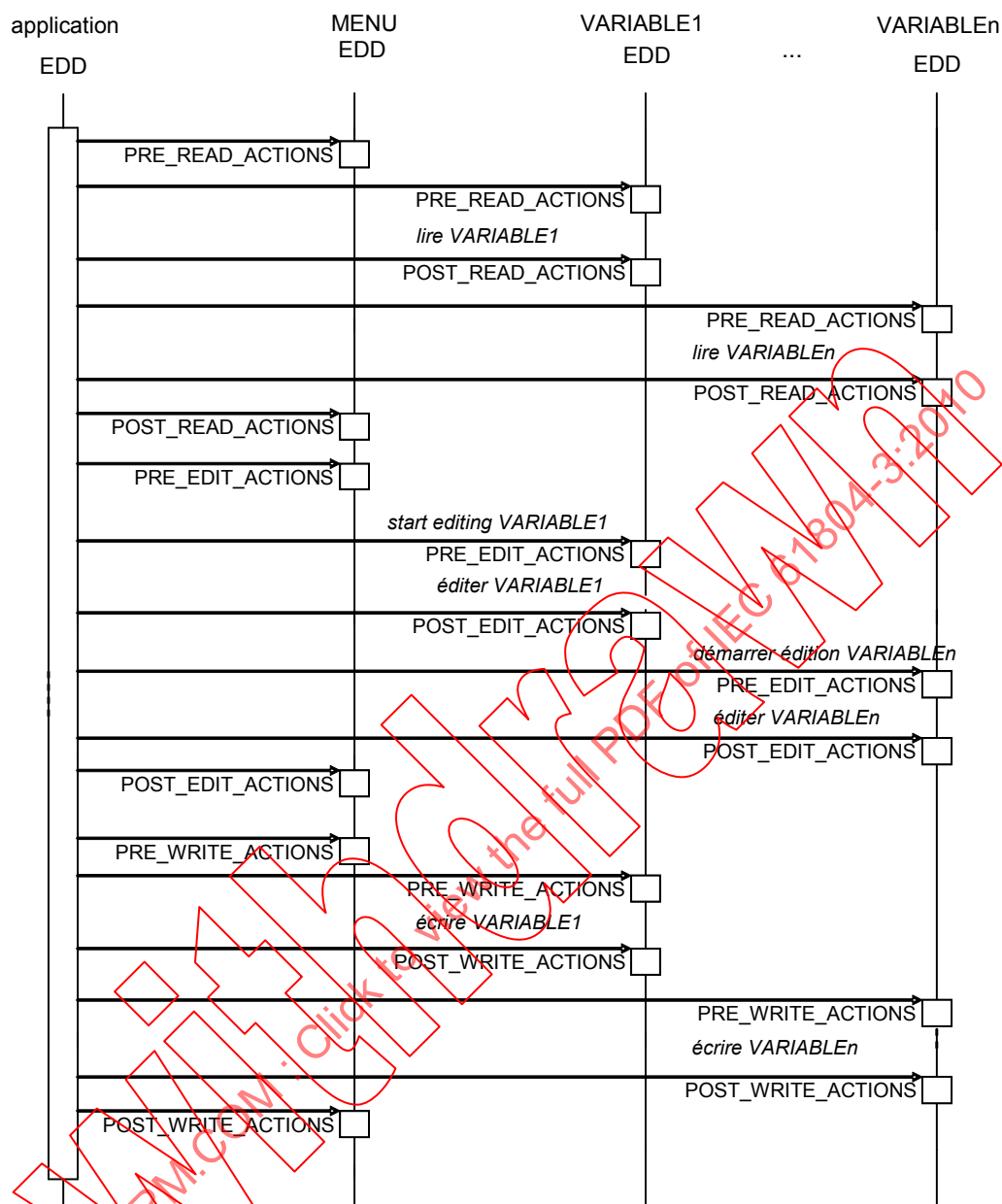


Figure 28 – Activation de MENU

IEC 2409/10

7.24 METHOD

7.24.1 Structure générale

Objet

Une METHOD (méthode) est utilisée pour définir un sous-programme qui est exécuté dans l'application EDD.

NOTE Une METHOD est utilisée pour spécifier des contrôles de cohérence, des algorithmes de conversion entre des variables de l'EDD, des accusés de réception d'utilisateur ou des interactions de dispositif spécifiques.

Structure lexicale

```
METHOD identifiant [(parameter-type parameter-name)+] [DEFINITION, ACCESS, CLASS, HELP, LABEL, TYPE, VALIDITY]
```

Les attributs de METHOD sont spécifiés dans le Tableau 143.

Tableau 143 – Attributs de METHOD

Usage	Attribut	Description
m	DEFINITION	Élément lexical (voir 7.36.4).
o	ACCESS	Élément lexical (voir 7.24.2.1).
o	CLASS	Élément lexical (voir 7.24.2.2).
o	HELP	Élément lexical (voir 7.36.8).
o	LABEL	Élément lexical (voir 7.36.9).
o	TYPE	Élément lexical (voir 7.24.2.3).
o	VALIDITY	Élément lexical (voir 7.36.16).
o	parameter-type	Type du paramètre (voir Tableau 134).
o	parameter-name	Nom du paramètre sous la forme d'un identifiant.

7.24.2 Attributs spécifiques

7.24.2.1 ACCESS

Objet

L'attribut ACCESS (accès) spécifie si les valeurs de variable stockées dans le dispositif ou l'ensemble de données hors ligne sont utilisées dans la DEFINITION de METHOD. Si ACCESS n'est pas défini, la valeur par défaut est ONLINE.

Structure lexicale

ACCESS (OFFLINE, ONLINE) <exp>

Les attributs d'ACCESS sont spécifiés dans le Tableau 144.

Tableau 144 – Attributs d'ACCESS

Usage	Attribut	Description
s	OFFLINE	Les valeurs des variables utilisées dans la définition de METHOD sont lues ou écrites depuis/vers l'ensemble de données hors connexion.
s	ONLINE	Les valeurs des variables utilisées dans la définition de METHOD sont effectivement lues depuis le dispositif. Les changements de valeurs de VARIABLE sont écrits avec des Builtin.

7.24.2.2 CLASS

Objet

L'attribut CLASS (classe) spécifie l'effet d'une méthode sur un dispositif de terrain. Cet attribut est destiné à être utilisé par l'application EDD pour implémenter des niveaux d'autorisation et définit comment des METHOD sont présentées. Les classes de METHOD sont identiques aux classes de VARIABLE, sauf qu'elles ne doivent pas être égales à LOCAL.

Structure lexicale

CLASS [ALARM, ANALOG_OUTPUT, COMPUTATION, CONTAINED, DEVICE, DIAGNOSTIC, DISCRETE, DYNAMIC, FREQUENCY, HART, INPUT, LOCAL_DISPLAY, OUTPUT, OPERATE, SENSOR_CORRECTION, TUNE, SERVICE] +

Les attributs de CLASS sont spécifiés dans le Tableau 163.

7.24.2.3 TYPE

Objet

L'attribut TYPE spécifie le type de la valeur retournée.

Structure lexicale

TYPE [char, unsigned char, int, unsigned int, long, unsigned long, float, double, time_t, DD_STRING]

Les attributs de TYPE sont spécifiés dans le Tableau 145.

Tableau 145 – Attributs de TYPE

Usage	Attribut	Description
s	char	La METHOD retourne une valeur de type char.
s	unsigned char	La METHOD retourne une valeur de type caractère non signé.
s	short	La METHOD retourne une valeur de type court.
s	unsigned short	La METHOD retourne une valeur de type court non signé.
s	int	La METHOD retourne une valeur de type entier.
s	unsigned int	La METHOD retourne une valeur de type entier non signé.
s	long	La METHOD retourne une valeur de type long.
s	unsigned long	La METHOD retourne une valeur de type long non signé.
s	float	La METHOD retourne une valeur de type à virgule flottante.
s	double	La METHOD retourne une valeur de type à double précision.
s	time_t	La METHOD retourne une valeur de type time_t.
s	DD_STRING	La METHOD retourne une valeur de type DD_STRING.

7.25 PROGRAM

7.25.1 Structure générale

Objet

PROGRAM (programme) permet la commande à distance pour l'exécution d'un programme.

EXEMPLE Un dispositif peut télécharger un PROGRAM dans un domaine d'un autre dispositif en utilisant un service de téléchargement puis commander à distance le PROGRAM en émettant une demande de service. Il convient que des PROGRAM puissent être créés, supprimés, démarrés, arrêtés, etc.

Structure lexicale

PROGRAM identifier [ARGUMENTS, RESPONSE_CODES]

Les attributs de PROGRAM sont spécifiés dans le Tableau 146.

Tableau 146 – Attributs de PROGRAM

Usage	Attribut	Description
o	ARGUMENTS	Élément lexical (voir 7.25.2).
o	RESPONSE_CODES	Élément lexical (voir 7.36.13).

7.25.2 Attribut spécifique – ARGUMENTS

Objet

L'attribut ARGUMENTS peut être envoyé au programme pendant le démarrage et la reprise des opérations. Les arguments de PROGRAM sont décrits par l'attribut ARGUMENTS. Une chaîne d'octet contenant les valeurs de tous les arguments est envoyée à l'objet qui invoque le programme lorsqu'il est démarré ou repris par l'application EDD.

Structure lexicale

ARGUMENTS [(data-item)<exp>]+

L'attribut d'ARGUMENT est spécifié dans le Tableau 147.

Tableau 147 – Attributs d'ARGUMENTS

Usage	Attribut	Description
m	data-item	Constante entière non signée ou VARIABLE. Si une donnée est une constante entière non signée, la valeur de la constante apparaît à cette position dans le champ de données. Les données constantes occupent deux octets du champ de données et par conséquent, doivent être dans l'intervalle de 0 à 65535 inclus. Si une donnée est une VARIABLE, la valeur de la variable apparaît à cette position dans le champ de données. Si le champ de données du service PROGRAM est vide, les arguments peuvent être omis.

7.26 RECORD

Objet

Un RECORD (enregistrement) est un groupe logique de VARIABLE. Chaque élément dans le RECORD est une référence à une VARIABLE et peut être de différents types de données.

NOTE Un RECORD peut être transféré sous la forme d'un objet de communication.

Structure lexicale

RECORD identifier [MEMBERS, LABEL, HELP, RESPONSE_CODES, VALIDITY]

Les attributs de RECORD sont spécifiés dans le Tableau 148.

Tableau 148 – Attributs de RECORD

Usage	Attribut	Description
m	MEMBERS	Élément lexical (voir 7.36.10).
m	LABEL	Élément lexical (voir 7.36.9).
o	HELP	Élément lexical (voir 7.36.8).
o	RESPONSE_CODES	Élément lexical (voir 7.36.13).
o	VALIDITY	Élément lexical (voir 7.36.16).

7.27 REFERENCE_ARRAY

7.27.1 Structure générale

Objet

Un REFERENCE_ARRAY (matrice de référence) est un ensemble d'éléments du type d'élément spécifié (par exemple, VARIABLE ou MENU).

NOTE Les REFERENCE_ARRAY ne sont pas liés aux matrices de communication (type d'élément "VALUE_ARRAY"). Les matrices de communication sont des matrices de valeurs.

Structure lexicale

REFERENCE_ARRAY identifier [item-type, ELEMENTS, HELP, LABEL, VALIDITY]

Les attributs de REFERENCE_ARRAY sont spécifiés dans le Tableau 149.

Tableau 149 – Attributs de REFERENCE_ARRAY

Usage	Attribut	Description
m	item-type	Type d'élément spécifié dans l'attribut ELEMENTS. Doit être un des éléments de base répertoriés dans le Tableau 47.
m	ELEMENTS	Élément lexical (voir 7.27.2).
o	HELP	Élément lexical (voir 7.36.8).
o	LABEL	Élément lexical (voir 7.36.9).
o	VALIDITY	Élément lexical (voir 7.36.16).

7.27.2 Attribut spécifique – ELEMENTS

Objet

L'attribut ELEMENTS (éléments) spécifie une liste d'éléments. Il définit pour chaque élément la référence d'élément, son index associé, sa description et une aide facultatives. Les attributs de description et d'aide, s'ils sont utilisés, remplacent les spécifications correspondantes dans l'EDD elle-même.

Structure lexicale

ELEMENTS [integer, reference, description, help]<exp>]+

Les attributs d'ELEMENTS sont spécifiés dans le Tableau 150.

Tableau 150 – Attributs d'ELEMENTS

Usage	Attribut	Description
m	integer	Numéro par lequel l'élément peut être référencé. Chaque numéro doit être défini de façon unique dans une construction REFERENCE_ARRAY.
m	reference	Référence à une instance de data-item (donnée).
o	description	Brève description de l'élément.
o	help	Texte d'aide pour l'élément.

7.28 Relations

7.28.1 REFRESH

Objet

Une relation REFRESH (réactualiser) est utilisée pendant l'accès en ligne et spécifie un ensemble de VARIABLE qu'il convient de réactualiser (lire depuis le dispositif) chaque fois qu'un ensemble spécifié est modifié.

NOTE Une VARIABLE peut avoir une relation REFRESH avec elle-même, ce qui signifie que la variable doit être lue après écriture.

Structure lexicale

REFRESH identifier, [(cause-parameter)<exp>]+, [(effected-parameter)<exp>]+

Les attributs de REFRESH sont spécifiés dans le Tableau 151.

Tableau 151 – Attributs de REFRESH

Usage	Attribut	Description
m	cause-parameter	Liste d'instances de VARIABLE.
m	effectuated-parameter	Liste d'instances de VARIABLE, qui doivent être lues, si un des paramètres de cause a été modifié. Une VARIABLE de cette liste de paramètres effectués peut également être dans la liste de paramètres de cause d'une autre relation REFRESH.

7.28.2 UNIT**Objet**

Une relation UNIT (unité) spécifie une référence à une VARIABLE contenant un code d'unité et une liste de VARIABLE dépendantes. Lorsque la variable contenant le code d'unité est modifiée, la liste de VARIABLE/AXIS réalisé utilisant ce code d'unité doit être réactualisée. Lorsqu'un VARIABLE/AXIS réalisé est affiché, la valeur de son code d'unité doit également être affichée.

Structure lexicale

UNIT *identifier*, [*cause-parameter*]*<exp>*], [(*effectuated-parameter*)]*<exp>*]+

Les attributs d'UNIT sont spécifiés dans le Tableau 152.

Tableau 152 – Attributs d'UNIT

Usage	Attribut	Description
m	cause-parameter	Référence à une instance de VARIABLE qui contient le code d'unité.
m	effectuated-parameter	Liste d'instances de VARIABLE/AXIS utilisant ce code d'unité.

7.28.3 WRITE_AS_ONE**Objet**

Une relation WRITE_AS_ONE spécifie une liste de VARIABLE qui doit être modifiée en tant que groupe par l'application EDD. Cette relation est utilisée lorsqu'un dispositif requiert que des VARIABLE spécifiques soient analysées et modifiées simultanément pour un fonctionnement correct.

NOTE Cette relation ne signifie pas nécessairement que les variables soient écrites vers le dispositif simultanément.

Structure lexicale

WRITE_AS_ONE *identifier*, [(*reference*)]*<exp>*]+

L'attribut de WRITE_AS_ONE est spécifié dans le Tableau 153.

Tableau 153 – Attribut de WRITE_AS_ONE

Usage	Attribut	Description
m	reference	Liste d'instances de VARIABLE, VALUE_ARRAY et RECORD qui doivent être modifiées en tant que groupe avec les autres membres de la relation WRITE_AS_ONE.

7.29 RESPONSE_CODES

Objet

Les RESPONSE_CODES (codes de réponse) spécifient les valeurs qu'un dispositif peut retourner en tant qu'information d'erreur. Chaque VARIABLE, RECORD, VALUE_ARRAY, VARIABLE_LIST, PROGRAM, COMMAND ou DOMAIN peut avoir son propre ensemble de RESPONSE_CODES.

Structure lexicale

```
RESPONSE_CODES identifier, [(description, integer, [DATA_ENTRY_ERROR,
DATA_ENTRY_WARNING, MISC_ERROR, MISC_WARNING, MODE_ERROR, PROCESS_ERROR,
SUCCESS], help)<exp>]+
```

Les attributs de RESPONSE_CODES sont spécifiés dans le Tableau 154.

Tableau 154 – Attributs de RESPONSE_CODES

Usage	Attribut	Description
m	description	Chaîne affichée lorsque le RESPONSE_CODE est retourné par le dispositif.
m	integer	Valeur retournée du dispositif pour identifier le RESPONSE_CODE.
s	DATA ENTRY ERROR	Le service de couche application a été rejeté parce que les données envoyées ne sont pas valides.
s	DATA ENTRY WARNING	Le service de couche application a été accepté et traité avec une version des données envoyées légèrement modifiée.
s	MISC ERROR	Le service de couche application a été rejeté.
s	MISC WARNING	Le service de couche application a été accepté et traité comme spécifié et il existe des informations additionnelles, sans rapport avec le service de couche application, susceptibles d'intéresser l'utilisateur.
s	MODE ERROR	Le service de couche application a été rejeté parce que le dispositif de terrain était dans un mode dans lequel le service de couche application ne pouvait pas être exécuté.
s	PROCESS ERROR	Le service de couche application a été rejeté parce qu'un processus appliqué au dispositif de terrain est non valide.
s	SUCCESS	Le service de couche application a été accepté et traité tel que spécifié.
o	help	Texte d'aide pour l'élément.

7.30 SOURCE

7.30.1 Structure générale

Objet

Une SOURCE décrit une source de valeurs de données pour un CHART (diagramme).

Structure lexicale

```
SOURCE identifier [MEMBERS, EMPHASIS, EXIT_ACTIONS, HELP, INIT_ACTIONS,
LABEL, LINE_COLOR, LINE_TYPE, REFRESH_ACTIONS, VALIDITY, Y_AXIS]
```

Les attributs de SOURCE sont spécifiés dans le Tableau 155.

Tableau 155 – Attributs de SOURCE

Usage	Attribut	Description
m	MEMBERS	Élément lexical (voir 7.36.12).
o	EMPHASIS	Élément lexical (voir 7.36.5).

Usage	Attribut	Description
o	EXIT_ACTIONS	Élément lexical (voir 7.30.2.1).
o	HELP	Élément lexical (voir 7.36.8).
o	INIT_ACTIONS	Élément lexical (voir 7.30.2.2).
o	LABEL	Élément lexical (voir 7.36.9).
o	LINE_COLOR	Élément lexical (voir 7.36.10).
o	LINE_TYPE	Élément lexical (voir 7.36.11).
o	REFRESH_ACTIONS	Élément lexical (voir 7.30.2.3).
o	VALIDITY	Élément lexical (voir 7.36.16).
o	Y_AXIS	Élément lexical (voir 7.30.2.4).

7.30.2 Attributs spécifiques

7.30.2.1 EXIT_ACTIONS

Objet

L'attribut EXIT_ACTIONS (actions de sortie) spécifie les actions qui sont exécutées lorsque le CHART (diagramme) contenant la SOURCE est fermé.

Les METHOD spécifiées doivent être exécutées dans l'ordre dans lequel elles apparaissent. Si une METHOD se termine pour une raison imprévue, les METHOD restantes ne doivent pas être exécutées.

L'attribut EXIT_ACTIONS ne doit pas être exprimé en utilisant une instruction conditionnelle (IF, IF-ELSE, SELECT). Si un comportement conditionnel est requis, les méthodes appelées par l'action doivent être compatibles avec celui-ci.

Structure lexicale

EXIT_ACTIONS [reference]+, DEFINITION

Les attributs d'EXIT_ACTIONS sont spécifiés dans le Tableau 188.

7.30.2.2 INIT_ACTIONS

Objet

L'attribut INIT_ACTIONS spécifie des actions qui sont exécutées avant que la valeur de SOURCE soit initialement affichée par un CHART (diagramme).

Les METHOD spécifiées doivent être exécutées dans l'ordre dans lequel elles apparaissent. Si une méthode se termine pour une raison imprévue, les METHOD restantes ne doivent pas être exécutées. De plus, l'application EDD ne doit pas représenter graphiquement la valeur de SOURCE si une méthode se termine pour une raison imprévue.

L'attribut INIT_ACTIONS ne doit pas être exprimé en utilisant une instruction conditionnelle (IF, IF-ELSE, SELECT). Si un comportement conditionnel est requis, les méthodes appelées par l'action doivent être compatibles avec celui-ci.

Structure lexicale

INIT_ACTIONS [reference]+, DEFINITION

Les attributs d'INIT_ACTIONS sont spécifiés dans le Tableau 188.

7.30.2.3 REFRESH_ACTIONS

Objet

L'attribut REFRESH_ACTIONS spécifie les actions qui sont exécutées avant l'affichage de la valeur sur le CHART dans lequel la SOURCE est affichée.

Les METHOD spécifiées doivent être exécutées dans l'ordre dans lequel elles apparaissent. Si une méthode se termine pour une raison imprévue, les METHOD restantes ne doivent pas être exécutées.

L'attribut REFRESH_ACTIONS ne doit pas être exprimé en utilisant une instruction conditionnelle (IF, IF-ELSE, SELECT). Si un comportement conditionnel est requis, les méthodes appelées par l'action doivent être compatibles avec celui-ci.

Structure lexicale

REFRESH_ACTIONS [reference]+, DEFINITION

Les attributs de REFRESH_ACTIONS sont spécifiés dans le Tableau 188.

7.30.2.4 Y_AXIS

Objet

L'attribut Y_AXIS spécifie le mode d'affichage de l'axe Y de la SOURCE. Chaque SOURCE sur un CHART partage un axe X commun qui est basé sur le temps, mais chaque SOURCE sur un CHART a son propre Y_AXIS.

Structure lexicale

Y_AXIS (reference)<exp>

L'attribut d' Y_AXIS est spécifié dans le Tableau 156.

Tableau 156 –Attribut de Y_AXIS

Usage	Attribut	Description
m	reference	Référence à une instance d'AXIS.

7.31 TEMPLATE

7.31.1 Structure générale

Objet

TEMPLATE (modèle) décrit un ensemble de valeurs de paramètre qui est utilisé pour la configuration hors ligne par l'application EDD. Chaque TEMPLATE définit des valeurs par défaut pour une configuration spécifique à une application. L'attribut DEFAULT_VALUES définit les valeurs par défaut des paramètres.

Structure lexicale

TEMPLATE identifieur [DEFAULT_VALUES, HELP, LABEL]

Les attributs de TEMPLATE sont spécifiés dans le Tableau 157.

Tableau 157 – Attributs de TEMPLATE

Usage	Attribut	Description
m	DEFAULT_VALUES	Élément lexical (voir 7.31.2).
m	LABEL	Élément lexical (voir 7.36.9).
o	HELP	Élément lexical (voir 7.36.8).

7.31.2 Attribut spécifique – DEFAULT_VALUES**Objet**

L'attribut DEFAULT_VALUES spécifie le réglage par défaut pour les VARIABLE. La valeur à définir doit être constante.

Structure lexicale

DEFAULT_VALUES (reference, value)+

Les attributs de DEFAULT_VALUES sont spécifiés dans le Tableau 158.

Tableau 158 – Attributs de DEFAULT_VALUES

Usage	Attribut	Description
m	reference	Référence à une instance de VARIABLE, un membre d'une instance de RECORD, un élément d'une instance de VALUE_ARRAY.
m	value	Constante qui spécifie la valeur. Le type dépend du type de données spécifié avec l'attribut de référence.

Les paramètres de référencement de bloc sont conformes à un référencement multi-bloc.

7.32 VALUE_ARRAY**7.32.1 Structure générale****Objet**

Un VALUE_ARRAY (matrice de valeurs) est un groupe logique de valeurs. Chaque élément d'un VALUE_ARRAY doit être du même type de données et doit correspondre à une construction EDD. Un élément est référencé depuis un autre emplacement dans l'EDD via l'identifiant et l'index d'élément de VALUE_ARRAY.

NOTE Du point de vue de la communication, les éléments individuels du VALUE_ARRAY ne sont pas traités comme des variables individuelles mais simplement comme une valeur groupée.

Structure lexicale

VALUE_ARRAY identifier [LABEL, NUMBER_OF_ELEMENTS, TYPE, HELP, RESPONSE_CODES, VALIDITY]

Les attributs de VALUE_ARRAY sont spécifiés dans le Tableau 159.

Tableau 159 – Attributs de VALUE_ARRAY

Usage	Attribut	Description
m	LABEL	Élément lexical (voir 7.36.9).
m	NUMBER_OF_ELEMENTS	Élément lexical (voir 7.32.2.1).
m	TYPE	Élément lexical (voir 7.32.2.2).
o	HELP	Élément lexical (voir 7.36.8).
o	RESPONSE_CODES	Élément lexical (voir 7.36.13).

Usage	Attribut	Description
o	VALIDITY	Élément lexical (voir 7.36.16).

7.32.2 Attributs spécifiques

7.32.2.1 NUMBER_OF_ELEMENTS

Objet

L'attribut NUMBER_OF_ELEMENTS spécifie le nombre d'éléments dans le VALUE_ARRAY (entier supérieur à zéro).

Structure lexicale

NUMBER_OF_ELEMENTS (integer, expression)

Les attributs de NUMBER_OF_ELEMENTS sont spécifiés dans le Tableau 160.

Tableau 160 – Attributs de NUMBER_OF_ELEMENTS

Usage	Attribut	Description
s	integer	Nombre d'éléments d'un VALUE_ARRAY.
s	expression	Expression qui doit être égale à un entier non négatif qui est dans l'intervalle d'index de l'attribut NUMBER_OF_ELEMENTS (pour la description des expressions, voir 7.39.7).

7.32.2.2 TYPE

Objet

L'attribut TYPE est une référence à une instance de construction EDDL (par exemple, VARIABLE, COLLECTION). Tous les attributs de la construction EDDL s'appliquent à chacun des éléments de VALUE_ARRAY.

Structure lexicale

TYPE reference

L'attribut de TYPE est spécifié dans le Tableau 161.

Tableau 161 – Attribut de TYPE

Usage	Attribut	Description
m	reference	Référence à une instance de construction de base d'EDD.

7.33 VARIABLE

7.33.1 Structure générale

Objet

Une VARIABLE décrit des paramètres contenus dans un dispositif ou dans l'application EDD.

Structure lexicale

VARIABLE identifier, [CLASS, LABEL, TYPE, CONSTANT_UNIT, DEFAULT_VALUE, HANDLING, HELP, INITIAL_VALUE, POST_EDIT_ACTIONS, POST_READ_ACTIONS, POST_WRITE_ACTIONS, PRE_EDIT_ACTIONS, PRE_READ_ACTIONS, PRE_WRITE_ACTIONS, READ_TIMEOUT, REFRESH_ACTIONS, RESPONSE_CODES, VALIDITY, WRITE_TIMEOUT]

Les attributs de VARIABLE sont spécifiés dans le Tableau 162.

Tableau 162 –Attributs de VARIABLE

Usage	Attribut	Description
m	CLASS	Élément lexical (voir 7.33.2.1).
o	LABEL	Élément lexical (voir 7.36.9).
m	TYPE	Élément lexical (voir 7.33.2.2).
o	CONSTANT_UNIT	Élément lexical (voir 7.33.2.3).
o	DEFAULT_VALUE	Élément lexical (voir 7.33.2.4).
o	HANDLING	Élément lexical (voir 7.36.6).
o	HELP	Élément lexical (voir 7.36.8).
o	INITIAL_VALUE	Élément lexical (voir 7.33.2.5).
o	POST_EDIT_ACTIONS	Élément lexical (voir 7.33.2.6).
o	POST_READ_ACTIONS	Élément lexical (voir 7.33.2.7).
o	POST_WRITE_ACTIONS	Élément lexical (voir 7.33.2.8).
o	PRE_EDIT_ACTIONS	Élément lexical (voir 7.33.2.9).
o	PRE_READ_ACTIONS	Élément lexical (voir 7.33.2.10).
o	PRE_WRITE_ACTIONS	Élément lexical (voir 7.33.2.11).
o	READ_TIMEOUT	Élément lexical (voir 7.33.2.12).
o	REFRESH_ACTIONS	Élément lexical (voir 7.33.2.13).
o	RESPONSE_CODES	Élément lexical (voir 7.36.13).
o	STYLE	Élément lexical (voir 7.33.2.14).
o	VALIDITY	Élément lexical (voir 7.36.15).
o	WRITE_TIMEOUT	Élément lexical (voir 7.33.2.12).

7.33.2 Attributs spécifiques

7.33.2.1 CLASS

Objet

L'attribut CLASS (classe) spécifie comment la VARIABLE est utilisée par le dispositif et l'application EDD à des fins d'organisation et d'affichage.

Les combinaisons de classes ne sont pas toutes autorisées. Les restrictions sont définies dans les tables de profils (voir Annexe D).

NOTE La définition de CLASS est basée sur un modèle dans lequel un dispositif est constitué d'une pluralité de blocs fonctionnels différents. Chaque bloc a une entrée et produit une sortie. Des blocs peuvent être combinés de différentes façons pour produire différentes fonctionnalités.

Structure lexicale

CLASS [ALARM, ANALOG_INPUT, ANALOG_OUTPUT, COMPUTATION, CONTAINED, CORRECTION, DEVICE, DIAGNOSTIC, DIGITAL_INPUT, DIGITAL_OUTPUT, DISCRETE_INPUT, DISCRETE_OUTPUT, DYNAMIC, FREQUENCY_INPUT, FREQUENCY_OUTPUT, HART, INPUT, LOCAL, LOCAL_DISPLAY, OPERATE, OUTPUT, SERVICE, TEMPORARY, TUNE]+

Les attributs de CLASS sont spécifiés dans le Tableau 163.

Tableau 163 – Attributs de CLASS

Usage	Attribut	Description
s	ALARM	Cette VARIABLE contient des limites d'alarme.
s	ANALOG_INPUT	Cette VARIABLE fait partie d'un bloc d'entrée analogique.
s	ANALOG_OUTPUT	Cette VARIABLE fait partie d'un bloc de sortie analogique.
s	COMPUTATION	Cette VARIABLE fait partie d'un bloc de calcul.
s	CONTAINED	Cette VARIABLE représente les caractéristiques physiques du dispositif.
s	CORRECTION	Cette VARIABLE fait partie du bloc de correction.
s	DEVICE	Cette VARIABLE représente les caractéristiques physiques du dispositif.
s	DIAGNOSTIC	Cette VARIABLE indique le statut du dispositif.
s	DIGITAL_INPUT	Cette VARIABLE fait partie d'un bloc d'entrée numérique.
s	DIGITAL_OUTPUT	Cette VARIABLE fait partie d'un bloc de sortie numérique.
s	DISCRETE_INPUT	Cette VARIABLE fait partie d'un bloc d'entrée discret.
s	DISCRETE_OUTPUT	Cette VARIABLE fait partie d'un bloc de sortie discret.
s	DYNAMIC	Cette VARIABLE est modifiée par le dispositif sans stimulus provenant du réseau.
s	FREQUENCY_INPUT	Cette VARIABLE fait partie d'un bloc d'entrée de fréquence.
s	FREQUENCY_OUTPUT	Cette VARIABLE fait partie d'un bloc de sortie de fréquence.
s	HART	Cette VARIABLE fait partie du bloc HART qui caractérise l'interface HART. Il y a un seul bloc HART.
s	INPUT	Cette VARIABLE fait partie d'un bloc d'entrée. Un bloc d'entrée est un type particulier de bloc de calcul qui effectue des conversions d'unité, une mise à l'échelle et un amortissement. Le paramètre des paramètres de bloc d'entrée peut être déterminé par la sortie d'un autre bloc.
s	LOCAL	Cette VARIABLE est utilisée localement par l'application EDD. Les VARIABLE locales ne sont pas stockées dans un dispositif, mais elles peuvent être envoyées à un dispositif.
s	LOCAL_DISPLAY	Cette VARIABLE fait partie du bloc d'affichage local. Un bloc d'affichage local contient les VARIABLE associées à l'interface locale (clavier, affichage, etc.) du dispositif.
s	OPERATE	Cette VARIABLE est utilisée pour contrôler le fonctionnement d'un bloc.
s	OPTIONAL	Cette VARIABLE peut ne pas être présente dans le dispositif.
s	OUTPUT	Cette VARIABLE fait partie du bloc de sortie. Les valeurs de VARIABLE de sortie peuvent être accessibles par un autre bloc d'entrée.
s	SERVICE	Cette VARIABLE est utilisée lors de la conduite de maintenance de routine.
s	TEMPORARY	Cette VARIABLE est initialisée à sa DEFAULT_VALUE (valeur par défaut) dans chaque session d'application EDD. Les variables LOCAL peuvent être rendues persistantes par l'application EDD; les variables TEMPORARY (temporaire) ne sont jamais persistantes.
s	TUNE	Cette VARIABLE est utilisée pour régler l'algorithme d'un bloc.

7.33.2.2 TYPE**7.33.2.2.1 Structure générale****Objet**

L'attribut TYPE décrit le format des valeurs de VARIABLE. Si TYPE est spécifié sans DEFAULT_VALUE (valeur par défaut) et INITIAL_VALUE (valeur initiale), la colonne "Initial value" dans le Tableau 164 présente les valeurs initiales.

Structure lexicale

TYPE [DOUBLE, FLOAT, INTEGER, UNSIGNED_INTEGER, DATE, DATE_AND_TIME, DURATION, TIME, TIME_VALUE, BIT_ENUMERATED, ENUMERATED, INDEX, OBJECT_REFERENCE, ASCII, BITSTRING, EUC, OCTET, PACKED_ASCII, PASSWORD, VISIBLE]

Les attributs de TYPE sont spécifiés dans le Tableau 164.

Tableau 164 – Attributs de TYPE

Usage	Attribut	Description	Valeur initiale
s	DOUBLE	Type arithmétique (voir 7.33.2.2.2.1).	0,0
s	FLOAT	Type arithmétique (voir 7.33.2.2.2.1).	0,0
s	INTEGER	Type arithmétique (voir 7.33.2.2.2.1).	0
s	UNSIGNED_INTEGER	Type arithmétique (voir 7.33.2.2.2.1).	0
s	DATE	Type de date/heure (voir 7.33.2.2.2.2).	1-Jan-1900
s	DATE_AND_TIME	Type de date/heure (voir 7.33.2.2.2.2).	1-Jan-1900 00:00:00
s	DURATION	Type de date/heure (voir 7.33.2.2.2.2).	0 jour, 0 heure, 0 minute, 0 seconde
s	TEMPS	Type de date/heure (voir 7.33.2.2.2.2).	00:00:00
s	TIME_VALUE(8)	Type de date/heure (voir 7.33.2.2.2.2).	1-Jan-1972 00:00:00
s	TIME_VALUE(4)	Type de date/heure (voir 7.33.2.2.2.2).	00:00:00
s	BIT_ENUMERATED	Type d'énumération (voir 7.33.2.2.2.3).	0
s	ENUMERATED	Type d'énumération (voir 7.33.2.2.2.4).	Valeur de la première énumération.
s	INDEX	Type d'index (voir 7.33.2.2.2.5).	Premier élément dans la matrice de référence associée.
s	OBJECT_REFERENCE	Référence d'objet (voir 7.33.2.2.2.6).	NULL
s	ASCII	Type de chaîne (voir 7.33.2.2.2.7).	Chaque caractère correspond à un espace.
s	BITSTRING	Type de chaîne (voir 7.33.2.2.2.7).	Chaque caractère correspond à un espace.
s	EUC	Type de chaîne (voir 7.33.2.2.2.7).	Chaque caractère correspond à un espace.
s	OCTET	Type de chaîne (voir 7.33.2.2.2.7).	Chaque caractère correspond à un espace.
s	PACKED_ASCII	Type de chaîne (voir 7.33.2.2.2.7).	Chaque caractère correspond à un point d'interrogation (?).
s	PASSWORD	Type de chaîne (voir 7.33.2.2.2.7).	Chaque caractère correspond à un espace.
s	VISIBLE	Type de chaîne (voir 7.33.2.2.2.7).	Chaque caractère correspond à un espace.

7.33.2.2.2 Attributs spécifiques

7.33.2.2.2.1 DOUBLE, FLOAT, INTEGER, UNSIGNED_INTEGER

Objet

Les VARIABLE de TYPE FLOAT et DOUBLE sont des nombres à virgule flottante au format de base simple précision et au format de base double précision, comme défini dans l'IEEE 754, respectivement. Les VARIABLE de TYPE INTEGER et UNSIGNED_INTEGER sont respectivement des nombres entiers signés et non signés.

Ces types sont appelés collectivement types arithmétiques.

Structure lexicale — DOUBLE

```
DOUBLE [DEFAULT_VALUE, DISPLAY_FORMAT, EDIT_FORMAT, INITIAL_VALUE,
[MAX_VALUE, m]*, [MIN_VALUE, n]*, SCALING_FACTOR], [(description, help,
value)<exp>]*
```

Structure lexicale — FLOAT

```
FLOAT [DEFAULT_VALUE, DISPLAY_FORMAT, EDIT_FORMAT, INITIAL_VALUE,
[MAX_VALUE, m]*, [MIN_VALUE, n]*, SCALING_FACTOR], [(description, help,
value)<exp>]*
```

Structure lexicale — INTEGER

```
INTEGER [DEFAULT_VALUE, DISPLAY_FORMAT, EDIT_FORMAT, INITIAL_VALUE,
[MAX_VALUE, m]*, [MIN_VALUE, n]*, SCALING_FACTOR, size], [description,
help, value]*
```

Structure lexicale — UNSIGNED_INTEGER

```
UNSIGNED_INTEGER [DEFAULT_VALUE, DISPLAY_FORMAT, EDIT_FORMAT,
INITIAL_VALUE, [MAX_VALUE, m]*, [MIN_VALUE, n]*, SCALING_FACTOR, size],
[(description, help, value)<exp>]*
```

Les attributs des types arithmétiques sont spécifiés dans le Tableau 165.

Tableau 165 – Attributs de DOUBLE, FLOAT, INTEGER, UNSIGNED_INTEGER

Usage	Attribut	Description
o	DEFAULT_VALUE	Objet L'attribut DEFAULT_VALUE spécifie le réglage par défaut pour la VARIABLE. La DEFAULT_VALUE est disponible pour tous les types EDDL. Si aucune INITIAL_VALUE n'est définie, la DEFAULT_VALUE est utilisée en tant qu'INITIAL_VALUE. La DEFAULT_VALUE peut également être générée par une expression (voir 7.40). Structure lexicale DEFAULT_VALUE (double)<exp> DEFAULT_VALUE (float)<exp> DEFAULT_VALUE (integer)<exp> DEFAULT_VALUE (unsigned_integer)<exp> DEFAULT_VALUE (expression)<exp>
o	description	Brève description de l'élément.
o	DISPLAY_FORMAT	Objet Un attribut DISPLAY_FORMAT spécifie comment la valeur de la variable est affichée. Les chaînes DISPLAY_FORMAT sont des spécificateurs de conversion des fonctions ANSI C (voir l'ISO/CEI 9899) printf et scanf, respectivement. Structure lexicale DISPLAY_FORMAT (string)<exp>
o	EDIT_FORMAT	Objet Un attribut EDIT_FORMAT spécifie le format pour l'édition. Les chaînes

Usage	Attribut	Description
		EDIT_FORMAT sont des spécificateurs de conversion des fonctions ANSI C (voir l'ISO/CEI 9899) printf et scanf, respectivement. Structure lexicale EDIT_FORMAT (string)<exp>
o	help	spécifie le texte d'aide pour l'élément.
o	INITIAL_VALUE	Objet Lorsqu'un dispositif est instancié pour la première fois, l'INITIAL_VALUE d'une VARIABLE est affichée. Il est fréquent que la DEFAULT_VALUE soit conditionnée et dépende d'autres paramètres. Dans ce cas, l'INITIAL_VALUE est utilisée pour reconstituer les réglages par défaut. L'INITIAL_VALUE est disponible pour tous les types et ne doit pas être conditionnée. Structure lexicale INITIAL_VALUE (double) INITIAL_VALUE (float) INITIAL_VALUE (integer) INITIAL_VALUE (unsigned_integer)
o	MAX_VALUE	Objet L'attribut MAX_VALUE spécifie la limite supérieure de valeurs auxquelles une VARIABLE doit être réglée. Si la variable est CLASS DYNAMIC, il convient que le dispositif soit réglé à une valeur de la variable comprise à l'intérieur de l'intervalle spécifié par ses valeurs minimales et maximales. Une VARIABLE peut avoir plus d'une valeur maximale, par exemple, pour spécifier plus d'un intervalle de validité. Lorsqu'il y a plusieurs valeurs maximales, un entier additionnel est utilisé pour numéroté chaque MAX_VALUE. Chaque MAX_VALUE doit avoir une MIN_VALUE équivalente. L'intervalle spécifié avec une paire de MIN_VALUE et MAX_VALUE ne peut pas chevaucher un autre intervalle spécifié. Si une MIN_VALUE/MAX_VALUE n'a pas de MAX_VALUE/MIN_VALUE équivalente, la limite supérieure/inférieure est non limitée et il n'est pas nécessaire de la spécifier. La MAX_VALUE peut également être générée par une expression (voir 7.40). Structure lexicale MAX_VALUE (double, integer)<exp> MAX_VALUE (float, integer)<exp> MAX_VALUE (integer, integer)<exp> MAX_VALUE (unsigned_integer, integer)<exp> MAX_VALUE (expression)<exp>
c	m	Entier croissant commençant à 1, qui identifie la MAX_VALUE si il y en plus d'une.
	MIN_VALUE	Objet L'attribut MIN_VALUE spécifie la limite inférieure de valeurs auxquelles une VARIABLE doit être réglée. Si la variable est CLASS DYNAMIC, il convient que le dispositif soit réglé à une valeur de la VARIABLE comprise dans l'intervalle spécifié par ses valeurs minimale et maximale. Une VARIABLE peut avoir plus d'une valeur minimale, par exemple, pour spécifier plus d'un intervalle de validité. Lorsqu'il existe plusieurs valeurs minimales, un entier additionnel est utilisé pour numéroté chaque MIN_VALUE. Chaque MIN_VALUE doit avoir une MAX_VALUE équivalente. L'intervalle spécifié avec une paire de MIN_VALUE et MAX_VALUE peut ne pas chevaucher un autre intervalle spécifié. Si une MIN_VALUE/MAX_VALUE n'a pas de MAX_VALUE/MIN_VALUE associée, la limite supérieure/inférieure est non limitée et il n'est pas nécessaire de la spécifier. La MIN_VALUE peut également être construite par une expression (voir 7.40). Structure lexicale MIN_VALUE (double, integer)<exp> MIN_VALUE (float, integer)<exp> MIN_VALUE (integer, integer)<exp> MIN_VALUE (unsigned_integer, integer)<exp> MIN_VALUE (expression)<exp>

Usage	Attribut	Description
c	n	Entier croissant commençant à 1, qui identifie la MIN_VALUE si il y en a plus d'une.
o	SCALING_FACTOR	Objet L'attribut SCALING_FACTOR (facteur d'échelle) indique que la valeur affichée de la variable n'est pas la valeur retournée par un dispositif. La valeur affichée est la valeur retournée par un dispositif multipliée par un facteur. Par conséquent, le processeur EDDL doit multiplier la valeur de la variable retournée par un dispositif avec son SCALING_FACTOR avant qu'elle soit affichée (ou avant qu'elle soit utilisée autrement). Le SCALING_FACTOR ne doit pas être égal à zéro. Le SCALING_FACTOR peut également être généré par une expression (voir 7.40). Le SCALING_FACTOR est appliqué pour l'affichage uniquement; la valeur de la variable, lorsqu'elle est assignée à une variable locale de méthode, est la valeur non mise à l'échelle retournée par le dispositif; les valeurs minimales et maximales, lorsqu'elles sont utilisées conjointement avec le facteur d'échelle, sont également des valeurs non mises à l'échelle. Structure lexicale SCALING_FACTOR (double)<exp> SCALING_FACTOR (expression)<exp>
o	size	Taille du type de données en octets. La taille est une constante entière supérieure à zéro et n'a pas de limite supérieure. Cette valeur est facultative. La valeur par défaut est 1.
o	value	Constante qui spécifie la valeur. Le type dépend du type de données spécifié avec l'attribut TYPE qui est facultatif, mais s'il est défini, une description d'attribut est requise. Des valeurs égales ne sont pas autorisées.

7.33.2.2.2.2 DATE, DATE_AND_TIME, DURATION, TIME, TIME_VALUE

Objet

Les VARIABLE de type de données DATE, DATE_AND_TIME, DURATION, TIME, TIME_VALUE sont utilisées pour spécifier des dates, heures, dates et heures, et des différences d'heure:

- le type de données DATE est constitué d'une date calendaire;
- le type de données DATE_AND_TIME (date/heure) est constitué d'une date calendaire et d'une heure;
- le type de données DURATION (durée) est une différence d'heure qui est constituée d'un temps en ms et d'un compteur de jours;
- le type de données TIME (heure) est constitué d'une heure et d'une date facultative;
- le type de données TIME_VALUE (temps) est utilisé pour représenter des différences d'heure ou l'heure du jour avec la précision requise pour la synchronisation d'horloge d'application.

Structure lexicale — DATE

DATE spécifie que la VARIABLE est un entier codé sur trois octets. Le codage des octets est spécifié en D.5.2.

DATE [DEFAULT_VALUE, INITIAL_VALUE]

Structure lexicale — DATE_AND_TIME

DATE_AND_TIME spécifie que la VARIABLE est un entier codé sur sept octets. Le codage des octets est spécifié en D.5.3.

DATE_AND_TIME [DEFAULT_VALUE, INITIAL_VALUE]

Structure lexicale — DURATION

DURATION spécifie que la VARIABLE est un entier codé sur 6 octets. Les quatre premiers bits doivent avoir la valeur zéro. Les bits 0 à 27 représentent l'heure en ms. Le nombre de jours est une valeur binaire de 16 bits. Le codage des octets est spécifié en D.5.4.

DURATION [DEFAULT_VALUE, INITIAL_VALUE]

Structure lexicale — TIME

TIME spécifie que la VARIABLE est un entier codé sur six octets. L'heure est exprimée en ms à partir de minuit. A minuit, le compteur recommence à zéro.

La date est indiquée en jours par rapport au 01-01-1984. Le premier jour de janvier 1984, le comptage de date commence à la valeur zéro. Le codage des octets est spécifié en D.5.5.

TIME [DEFAULT_VALUE, INITIAL_VALUE]

Structure lexicale — TIME_VALUE

TIME_VALUE spécifie que la VARIABLE est un entier codé sur huit octets. C'est un nombre à décimale fixe, qui est le temps exprimé en multiples de 1/32 ms. Le codage des octets est spécifié en D.5.6.

TIME_VALUE [DEFAULT_VALUE, DISPLAY_FORMAT, EDIT_FORMAT, INITIAL_VALUE, size, TIME_FORMAT, TIME_SCALE]

Les attributs des types DATE, DATE_AND_TIME, DURATION, TIME, TIME_VALUE sont spécifiés dans le Tableau 166.

Tableau 166 – Attributs de DATE, DATE_AND_TIME, DURATION, TIME, TIME_VALUE

Usage	Attribut	Description
o	DEFAULT_VALUE	Objet L'attribut DEFAULT_VALUE spécifie le réglage par défaut pour la variable. La DEFAULT_VALUE est disponible pour tous les types EDDL. Si aucune INITIAL_VALUE n'est définie, la DEFAULT_VALUE est utilisée en tant qu'INITIAL_VALUE. La DEFAULT_VALUE peut également être générée par une expression (voir 7.40). Structure lexicale DEFAULT_VALUE (integer)<exp> DEFAULT_VALUE (unsigned_integer)<exp> DEFAULT_VALUE (expression)<exp>
o	DISPLAY_FORMAT	Voir Tableau 165.
o	EDIT_FORMAT	Voir Tableau 165.
o	INITIAL_VALUE	Objet Lorsqu'un dispositif est instancié pour la première fois, l'INITIAL_VALUE d'une VARIABLE est affichée. La DEFAULT_VALUE est conditionnée et dépend souvent d'autres paramètres. Dans ce cas, l'INITIAL_VALUE est utilisée pour reconstituer les réglages par défaut. L'INITIAL_VALUE est disponible pour tous les types et ne doit pas être conditionnée. Lexical structure INITIAL_VALUE (integer) INITIAL_VALUE (unsigned_integer)
o	size	Taille du type de données en octets. La taille doit être de 4 ou 8. La valeur par défaut est 4. Cet attribut est applicable uniquement à TIME_VALUE.
o	TIME_FORMAT	Objet L'attribut TIME_FORMAT spécifie comment la variable est affichée et éditée. Les chaînes TIME_FORMAT peuvent comprendre les spécificateurs de format %H, %I, %M, %p, %R, %S, et %T spécifiés par la fonction ANSI C (voir l'ISO/CEI 9899). Lorsque TIME_FORMAT est spécifié, DISPLAY_FORMAT et EDIT_FORMAT ne doivent pas être

Usage	Attribut	Description
		spécifiés. Structure lexicale TIME_FORMAT (string)<exp>
o	TIME_SCALE	Objet L'attribut TIME_SCALE spécifie comment la variable est mise à l'échelle: SECONDS (secondes), MINUTES (minutes) ou HOURS (heures). Lorsqu'il est présent, TIME_SCALE déduit un facteur d'échelle de 1/32000, 1/1920000, ou 1/115200000 et une unité constante de secondes ("s"), minutes ("min"), ou heures ("h"), respectivement. Lorsque TIME_SCALE est spécifié, DISPLAY_FORMAT et EDIT_FORMAT peuvent également être spécifiés. Cet attribut est applicable uniquement à TIME_VALUE Structure lexicale TIME_SCALE (SECONDS, MINUTES, HOURS)<exp>

7.33.2.2.2.3 BIT_ENUMERATED

Objet

Les variables de type de données BIT_ENUMERATED sont des constantes entières non signées qui identifient un bit de la valeur de variable avec sa position.

Structure lexicale

BIT_ENUMERATED [(description, value, actions, function, help, status-class*)<exp>]+, [DEFAULT_VALUE, INITIAL_VALUE, size]

Les attributs du type BIT_ENUMERATED sont spécifiés dans le Tableau 167.

Tableau 167 – Attributs de BIT_ENUMERATED

Usage	Attribut	Description
m	description	Voir Tableau 165.
m	value	Entier non signé qui spécifie une position de bit. Si la valeur de la variable affecte aux positions de bit spécifiées un 1 logique, la description associée est affichée. Des valeurs égales ne sont pas autorisées.
o	actions	Actions exécutées lorsque le bit est défini.
o	function	Classe fonctionnelle du bit. La classe fonctionnelle d'un bit est la même que la CLASS d'une VARIABLE. Si aucune fonction n'est spécifiée, la valeur de la classe de fonction est égale par défaut à la classe de la variable. Par conséquent, si tous les bits ont la même fonction, la CLASS de la variable doit être spécifiée.
o	help	Voir Tableau 165.
o	size	Voir Tableau 165.
o	status-class	Signification du bit. Un bit de statut peut appartenir à plus d'une status-class (classe de statut, voir Tableau 168). Si la variable n'est pas un octet de statut, les bits n'ont pas la classe de statut.
o	DEFAULT_VALUE	Objet L'attribut DEFAULT_VALUE spécifie le réglage par défaut pour la variable. La DEFAULT_VALUE est disponible pour tous les types EDDL. Si aucune INITIAL_VALUE n'est définie, la DEFAULT_VALUE est utilisée en tant qu'INITIAL_VALUE. La DEFAULT_VALUE peut également être générée par une expression (voir 7.40). Structure lexicale DEFAULT_VALUE (integer)<exp> DEFAULT_VALUE (unsigned_integer)<exp> DEFAULT_VALUE (expression)<exp>

Usage	Attribut	Description
o	INITIAL_VALUE	Objet Lorsqu'un dispositif est instancié pour la première fois, l'INITIAL_VALUE d'une VARIABLE est affichée. Il est fréquent que la DEFAULT_VALUE soit conditionnée et dépende d'autres paramètres. Dans ce cas, l'INITIAL_VALUE est utilisée pour reconstituer les réglages par défaut. L'INITIAL_VALUE est disponible pour tous les types et ne doit pas être conditionnée. Structure lexicale INITIAL_VALUE (integer) INITIAL_VALUE (unsigned_integer)

Le Tableau 168 présente les status-class (classe de statut) pour une VARIABLE de type BIT_ENUMERATED.

Tableau 168 – Attributs de status-class

Usage	Attribut	Description
m	DATA	Configuration de données non valide.
m	HARDWARE	Défaillance du matériel.
m	MISC	Conditions diverses.
m	MODE	Le dispositif est dans un mode particulier.
m	PROCESS	Problème avec le processus connecté au dispositif de terrain.
m	SOFTWARE	Défaillance du logiciel.
m	CORRECTABLE	Le bit peut être effacé par l'application EDD ou le processus connecté.
m	SELF_CORRECTING	Le bit sera réinitialisé sans intervention supplémentaire.
m	UNCORRECTABLE	Le bit n'est ni auto-corrigeable, ni corrigeable.
m	EVENT	Événement unique.
m	STATE	Le dispositif est dans un état particulier.
m	COMM_ERROR	Défaillance de la communication dans l'autre dispositif.
o	IGNORE_IN_HANDHELD	Il convient qu'un bit soit ignoré dans des dispositifs de communication portatifs.
o	IGNORE_IN_TEMPORARY_MASTER	Il convient qu'un bit soit ignoré dans des dispositifs maîtres temporaires.
m	MORE	Des informations de statut additionnelles sont disponibles en provenance du dispositif.
o	ALL	Affecte toutes les sorties (voir Tableau 169).
o	AO	Affecte une des sorties analogiques (voir Tableau 169).
m	BAD	La sortie n'est pas fiable, il convient de ne pas l'utiliser pour la commande (voir Tableau 169).
o	DV	Affecte une des variables dynamiques (voir Tableau 169).
o	TV	Affecte une des variables d'émetteur (voir Tableau 169).
m	DETAIL	Le bit est répété ailleurs dans un bit de classe SUMMARY (résumé).
m	SUMMARY	Le bit est une combinaison logique d'autres bits de classe DETAIL (détail).
o	INFO	Le bit donne des informations sur le statut du processus ou du dispositif.
o	WARNING	Le bit donne un avertissement sur le statut du processus ou du dispositif.
o	IGNORE_IN_HOST	Le bit doit être ignoré dans un hôte fixe, raccordé en permanence.

Les status-class (classe de statut) ALL, AO, DV et TV donnent des informations de statut supplémentaires sur la sortie d'un dispositif.

Structure lexicale — ALL

ALL [AUTO_BAD, AUTO_GOOD, MANUAL_BAD, MANUAL_GOOD]

Structure lexicale — AO

AO [AUTO_BAD, AUTO_GOOD, integer, MANUAL_BAD, MANUAL_GOOD]

Structure lexicale — DV

DV [AUTO_BAD, AUTO_GOOD, integer, MANUAL_BAD, MANUAL_GOOD]

Structure lexicale — TV

TV [AUTO_BAD, AUTO_GOOD, integer, MANUAL_BAD, MANUAL_GOOD]

Les attributs sont spécifiés dans le Tableau 169.

Tableau 169 – Attributs ALL, AO, DV, TV

Usage	Attribut	Description
s	AUTO_BAD	La valeur provient d'un processus d'application et n'est pas valide.
s	AUTO_GOOD	La valeur provient d'un processus d'application et n'est pas valide.
m	integer	La valeur de VARIABLE provient du dispositif.
s	MANUAL_BAD	La valeur ne provient pas d'un processus d'application et n'est pas valide.
s	MANUAL_GOOD	La valeur ne provient pas d'un processus d'application et n'est pas valide.

7.33.2.2.2.4 ENUMERATED**Objet**

Une VARIABLE de type ENUMERATED est un entier non signé. Une liste associée de triplets (entier, description, aide) spécifie les valeurs possibles de la variable ainsi qu'une description et une aide facultative.

Structure lexicale

ENUMERATED [(description, value, help)<exp>]+, [DEFAULT_VALUE, INITIAL_VALUE, size]

Les attributs du type ENUMERATED sont spécifiés dans le Tableau 170.

Tableau 170 – Attributs de types énumérés

Usage	Attribut	Description
m	description	Voir Tableau 165.
m	value	Constante qui spécifie la valeur de VARIABLE. Le type est un entier non signé. La liste d'énumération est facultative mais si elle est définie, une valeur et une description sont requises. Des valeurs égales ne sont pas autorisées.
o	DEFAULT_VALUE	Objet La DEFAULT_VALUE spécifie la valeur par défaut pour la variable. La DEFAULT_VALUE est disponible pour tous les types EDDL. Si aucune INITIAL_VALUE n'est définie, la DEFAULT_VALUE est utilisée en tant qu'INITIAL_VALUE. La DEFAULT_VALUE peut également être générée par une expression, voir 7.40). Structure lexicale DEFAULT_VALUE (unsigned_integer)<exp> DEFAULT_VALUE (expression)<exp>
o	help	Voir Tableau 165.
o	INITIAL_VALUE	Objet Lorsqu'un dispositif est instancié pour la première fois, l'INITIAL_VALUE d'une VARIABLE est affichée. Il est fréquent que la DEFAULT_VALUE soit conditionnée et dépende d'autres paramètres. Dans ce cas,

Usage	Attribut	Description
		L'INITIAL_VALUE est utilisée pour reconstituer les paramètres par défaut. L'INITIAL_VALUE est disponible pour tous les types et ne doit pas être conditionnée. Structure lexicale INITIAL_VALUE (unsigned_integer)
o	size	Voir Tableau 165.

7.33.2.2.2.5 INDEX

Objet

Une VARIABLE de TYPE INDEX est un entier non signé qui est utilisé pour référencer les éléments d'un REFERENCE_ARRAY (voir 7.27). L'attribut entier des ELEMENTS (voir 7.27.2) spécifie les valeurs autorisées de la VARIABLE.

NOTE Lorsqu'une VARIABLE INDEX est présentée à l'utilisateur, le texte associé aux index de la matrice doit être affiché, mais pas la valeur numérique de la variable.

Structure lexicale

INDEX (reference, size)<exp>

Les attributs du type d'index sont spécifiés dans le Tableau 171.

Tableau 171 – Attributs de type index

Usage	Attribut	Description
o	size	Taille de la variable en octets. Cette valeur est une constante entière supérieure à zéro et n'a pas de limite supérieure. La valeur par défaut est 1. La REFERENCE_ARRAY doit contenir uniquement des éléments qui ne dépassent pas la taille d'index de la variable.
m	reference	Référence à une instance de REFERENCE_ARRAY.

7.33.2.2.2.6 Types de référence d'objet

Objet

Les VARIABLE du type OBJECT_REFERENCE permettent d'accéder aux valeurs d'autres dispositifs.

Structure lexicale

OBJECT_REFERENCE DEFAULT_REFERENCE

Les attributs des types OBJECT_REFERENCE sont spécifiés dans le Tableau 172.

Tableau 172 – Attribut du type de référence d'objet

Usage	Attribut	Description
o	DEFAULT_REFERENCE	La VARIABLE est utilisée pour accéder aux valeurs d'autres dispositifs. Si aucune DEFAULT_REFERENCE n'est définie, la VARIABLE est initialisée avec SELF (voir Tableau 173).

Les attributs de DEFAULT-REFERENCE sont spécifiés dans le Tableau 173.

Tableau 173 – Attributs de DEFAULT_REFERENCE

Usage	Attribut	Description
s	CHILD	Navigue d'un niveau hiérarchique vers le bas.
s	FIRST	Navigue jusqu'au premier objet dans la hiérarchie.
s	LAST	Navigue jusqu'au dernier objet dans la hiérarchie.
s	NEXT	Navigue jusqu'à l'objet suivant de la même hiérarchie.
s	NULL	Référence d'objet vide.
s	PARENT	Navigue d'une hiérarchie de bas en haut.
s	PREV	Navigue vers l'objet précédent de la même hiérarchie.
s	reference	Référence à un dispositif projeté dans le réseau.
s	SELF	Retourne vers lui-même dans sa propre hiérarchie.

7.33.2.2.2.7 ASCII, BIT_STRING, EUC, PACKED_ASCII, OCTET, PASSWORD, VISIBLE**Objet**

Les variables de chaîne comprennent les types suivants.

- ASCII – utilisé pour spécifier une séquence de caractères du jeu de caractères ISO Latin-1.
- BITSTRING – Les variables de type de données BIT ENUMERATED sont des constantes de chaîne qui identifient la position d'un caractère de la valeur de VARIABLE par rapport à sa position.
- EUC (code d'unité étendu) – est le code interne pour des caractères spécifiques (par exemple, Chine: EUC-CN, Taiwan: EUC-TW). EUC est utilisé pour gérer les langues asiatiques (ISO/CEI 10646-1).
- Le processeur EDDL doit prendre en charge un schéma de codage dans lequel des caractères sont codés en utilisant un nombre variable d'octets. Typiquement, certains octets signalent le début d'un caractère et combien d'octets additionnels sont utilisés pour coder le caractère. Des jeux de caractères avec un grand nombre d'éléments sont souvent stockés en utilisant un schéma d'encapsulation.
- PACKED_ASCII – est un sous-ensemble d'ASCII obtenu en enlevant les deux bits les plus significatifs de chaque caractère ASCII (voir D.5.7).
- PASSWORD – est un type chaîne utilisé pour spécifier des chaînes des mots de passe. À l'exception du mode de présentation de la variable à l'utilisateur, les types de chaîne de mot de passe et ASCII sont identiques.
- VISIBLE – Cette chaîne est une séquence ordonnée de caractères du jeu de caractères de l'ISO/CEI 10646-1 et l'ISO/CEI 2375.
- OCTET – est utilisé pour spécifier une séquence de données binaires non formatées dont la définition est déterminée par l'implémentation du dispositif.

Structure lexicale — EUC

```
EUC [(string, description, help)<exp>]+, [size, DEFAULT_VALUE, INITIAL_VALUE]
```

Structure lexicale — ASCII

```
ASCII [(string, description, help)<exp>]+, [size, DEFAULT_VALUE, INITIAL_VALUE]
```

Structure lexicale – PACKED_ASCII

```
PACKED_ASCII [(string, description, help)<exp>]+, [size, DEFAULT_VALUE, INITIAL_VALUE]
```


Structure lexicale — PASSWORD

PASSWORD [(string, description, help)<exp>]+, [size, DEFAULT_VALUE, INITIAL_VALUE]

Structure lexicale — BITSTRING

BITSTRING [(string, description, help)<exp>]+, [size, DEFAULT_VALUE, INITIAL_VALUE]

Structure lexicale — VISIBLE

VISIBLE [(string, description, help)<exp>]+, [size, DEFAULT_VALUE, INITIAL_VALUE]

Structure lexicale — OCTET

OCTET [(string, description, help)<exp>]+, [size, DEFAULT_VALUE, INITIAL_VALUE]

Les attributs des types de chaîne sont spécifiés dans le Tableau 174.

Tableau 174 – Attributs de types de chaîne

Usage	Attribut	Description
o	size	Taille du type de données en octets. La taille est une constante entière supérieure à zéro et n'a pas de limite supérieure. Cette valeur n'est pas facultative; une taille doit être spécifiée.
o	DEFAULT_VALUE	Objet La DEFAULT_VALUE spécifie le réglage par défaut pour la variable. La DEFAULT_VALUE est disponible pour tous les types EDDL. Si aucune INITIAL_VALUE n'est définie, la DEFAULT_VALUE est utilisée en tant qu'INITIAL_VALUE. Structure lexicale DEFAULT_VALUE (string)<exp>
o	INITIAL_VALUE	Objet Lorsqu'un dispositif est instancié pour la première fois, l'INITIAL_VALUE d'une VARIABLE est affichée. Il est fréquent que la DEFAULT_VALUE soit conditionnée et dépende d'autres paramètres. Dans ce cas, l'INITIAL_VALUE est utilisée pour reconstituer les paramètres par défaut. L'INITIAL_VALUE est disponible pour tous les types et ne doit pas être conditionnée. Structure lexicale INITIAL_VALUE (string)
o	string	Constante de type chaîne. La liste d'énumération est facultative mais si elle est définie, une chaîne et une description sont requises. Des chaînes égales ne sont pas autorisées.
o	description	Voir Tableau 165.
o	help	Voir Tableau 165.

7.33.2.3 CONSTANT_UNIT**Objet**

L'attribut CONSTANT_UNIT (unité constante) est utilisé, si une VARIABLE a un code d'unité qui ne change jamais. CONSTANT_UNIT est spécifiée comme étant une chaîne qui sera affichée avec la valeur. Une VARIABLE sans unité constante ne comporte pas d'unités associées à celle-ci ou bien les unités ne sont pas constantes.

Structure lexicale

CONSTANT_UNIT (string)<exp>

L'attribut de CONSTANT_UNIT est spécifié dans le Tableau 175.

Tableau 175 – Attribut CONSTANT_UNIT

Usage	Attribut	Description
m	string	Chaîne de code d'unités à afficher.

7.33.2.4 DEFAULT_VALUE

Objet

L'attribut DEFAULT_VALUE (valeur par défaut) spécifie le réglage par défaut pour la variable. Si aucune INITIAL_VALUE n'est définie, la DEFAULT_VALUE est utilisée en tant qu'INITIAL_VALUE.

Structure lexicale

DEFAULT_VALUE (expression)<exp>

NOTE 1 Il faut que le type de l'expression soit cohérent avec le type de la variable.

NOTE 2 La DEFAULT_VALUE peut également être spécifiée via l'attribut TYPE de la variable.

L'attribut de DEFAULT_VALUE est spécifié dans le Tableau 176.

Tableau 176 – Attribut DEFAULT_VALUE

Usage	Attribut	Description
m	expression	La valeur par défaut de la variable.

7.33.2.5 INITIAL_VALUE

Objet

L'attribut INITIAL_VALUE (valeur initiale) spécifie la valeur d'une VARIABLE qui est affichée lorsqu'un dispositif est instancié pour la première fois. Il est fréquent que la DEFAULT_VALUE soit conditionnée et dépende d'autres paramètres. Dans ce cas, l'INITIAL_VALUE est utilisée pour reconstituer les paramètres par défaut. L'INITIAL_VALUE est disponible pour tous les types et ne doit pas être conditionnée.

Structure lexicale

Expression d'INITIAL_VALUE

NOTE 1 Il faut que le type de l'expression soit cohérent avec le type de la variable.

NOTE 2 L'INITIAL_VALUE peut également être spécifiée via l'attribut TYPE de la variable.

L'attribut d'INITIAL_VALUE est spécifié dans le Tableau 177.

Tableau 177 – Attribut d'INITIAL_VALUE

Usage	Attribut	Description
m	expression	Valeur initiale de la variable.

7.33.2.6 POST_EDIT_ACTIONS

Objet

L'attribut POST_EDIT_ACTIONS (actions post-édition) spécifie les METHOD qui doivent être exécutées après que l'utilisateur ait terminé d'éditer la VARIABLE. Les METHOD spécifiées doivent être exécutées dans l'ordre dans lequel elles apparaissent. Si une METHOD se termine pour une raison imprévue, les METHOD suivantes ne sont pas exécutées.

L'attribut POST_EDIT_ACTIONS ne doit pas être exprimé en utilisant une instruction conditionnelle (IF, IF-ELSE, SELECT). Si un comportement conditionnel est requis, les méthodes appelées par l'action doivent être compatibles avec celui-ci.

Structure lexicale

POST_EDIT_ACTIONS [reference]+, DEFINITION

Les attributs de POST_EDIT_ACTIONS sont spécifiés dans le Tableau 178.

Tableau 178 – Attributs de POST_EDIT_ACTIONS, PRE_EDIT_ACTIONS, POST_READ_ACTIONS, PRE_READ_ACTIONS, POST_WRITE_ACTIONS, PRE_WRITE_ACTIONS, REFRESH_ACTIONS

Usage	Attribut	Description
o	reference	Référence à une instance de METHOD ou une instance de METHOD avec arguments.
o	DEFINITION	Élément lexical (voir 7.36.1).

7.33.2.7 POST_READ_ACTIONS

Objet

L'attribut POST_READ_ACTIONS (actions post-lecture) spécifie les METHOD qui doivent être exécutées après que la variable ait été lue depuis le dispositif. Les METHOD spécifiées doivent être exécutées dans l'ordre dans lequel elles apparaissent. Si une METHOD se termine pour une raison imprévue, les METHOD suivantes ne sont pas exécutées.

L'attribut POST_READ_ACTIONS ne doit pas être exprimé en utilisant une instruction conditionnelle (IF, IF-ELSE, SELECT). Si un comportement conditionnel est requis, les méthodes appelées par l'action doivent être compatibles avec celui-ci.

Structure lexicale

POST_READ_ACTIONS [reference]+, DEFINITION

Les attributs de POST_READ_ACTIONS sont spécifiés dans le Tableau 178.

7.33.2.8 POST_WRITE_ACTIONS

Objet

L'attribut POST_WRITE_ACTIONS (actions post-écriture) spécifie les METHOD qui doivent être exécutées après que la variable ait été écrite vers le dispositif. Les METHOD spécifiées doivent être exécutées dans l'ordre dans lequel elles apparaissent. Si une METHOD se termine pour une raison imprévue, les METHOD suivantes ne sont pas exécutées.

L'attribut POST_WRITE_ACTIONS ne doit pas être exprimé en utilisant une instruction conditionnelle (IF, IF-ELSE, SELECT). Si un comportement conditionnel est requis, les méthodes appelées par l'action doivent être compatibles avec celui-ci.

Structure lexicale

POST_WRITE_ACTIONS [reference]+, DEFINITION

Les attributs de POST_WRITE_ACTIONS sont spécifiés dans le Tableau 178.

7.33.2.9 PRE_EDIT_ACTIONS**Objet**

L'attribut PRE_EDIT_ACTIONS (actions pré-édition) spécifie les METHOD qui doivent être exécutées immédiatement lorsque la variable va être éditée. Les METHOD spécifiées doivent être exécutées dans l'ordre dans lequel elles apparaissent. Si une METHOD se termine pour une raison imprévue, les METHOD suivantes ne sont pas exécutées.

L'attribut PRE_EDIT_ACTIONS ne doit pas être exprimé en utilisant une instruction conditionnelle (IF, IF-ELSE, SELECT). Si un comportement conditionnel est requis, les méthodes appelées par l'action doivent être compatibles avec celui-ci.

Structure lexicale

PRE_EDIT_ACTIONS [reference]+, DEFINITION

Les attributs de PRE_EDIT_ACTIONS sont spécifiés dans le Tableau 178.

7.33.2.10 PRE_READ_ACTIONS**Objet**

L'attribut PRE_READ_ACTIONS (actions pré-lecture) spécifie les METHOD qui doivent être exécutées avant que la variable soit lue. Les METHOD spécifiées doivent être exécutées dans l'ordre dans lequel elles apparaissent. Si une METHOD se termine pour une raison imprévue, les METHOD suivantes ne sont pas exécutées.

L'attribut PRE_READ_ACTIONS ne doit pas être exprimé en utilisant une instruction conditionnelle (IF, IF-ELSE, SELECT). Si un comportement conditionnel est requis, les méthodes appelées par l'action doivent être compatibles avec celui-ci.

Structure lexicale

PRE_READ_ACTIONS [reference]+, DEFINITION

Les attributs de PRE_READ_ACTIONS sont spécifiés dans le Tableau 178.

7.33.2.11 PRE_WRITE_ACTIONS**Objet**

L'attribut PRE_WRITE_ACTIONS (actions pré-écriture) spécifie les METHOD qui doivent être exécutées avant que la variable soit écrite dans le dispositif. Les METHOD spécifiées doivent être exécutées dans l'ordre dans lequel elles apparaissent. Si une METHOD se termine pour une raison imprévue, les METHOD suivantes ne sont pas exécutées.

L'attribut PRE_WRITE_ACTIONS ne doit pas être exprimé en utilisant une instruction conditionnelle (IF, IF-ELSE, SELECT). Si un comportement conditionnel est requis, les méthodes appelées par l'action doivent être compatibles avec celui-ci.

Structure lexicale

PRE_WRITE_ACTIONS [reference]+, DEFINITION

Les attributs de PRE_WRITE_ACTIONS sont spécifiés dans le Tableau 178.

7.33.2.12 READ_TIMEOUT, WRITE_TIMEOUT

Objet

L'attribut READ_TIMEOUT (délai d'attente de lecture) spécifie la durée, en ms, pendant laquelle l'application EDD doit attendre la VARIABLE retournée. De manière similaire, l'attribut WRITE_TIMEOUT (délai d'attente d'écriture) spécifie la durée, en ms, pendant laquelle une application EDD doit attendre la confirmation que la variable a été écrite avec succès dans le dispositif.

Par exemple, il convient qu'un WRITE_TIMEOUT indique la durée nécessaire pour qu'un dispositif stocke la valeur d'une variable.

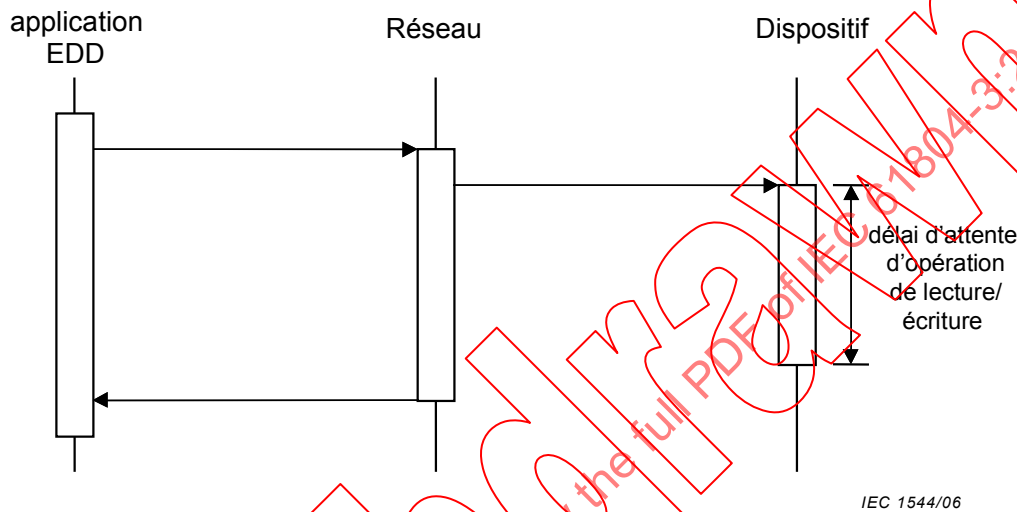


Figure 29 – Délai d'attente d'opération de lecture/écriture

Structure lexicale

```
READ_TIMEOUT (integer)<exp>
WRITE_TIMEOUT (integer)<exp>
```

L'attribut de READ_TIMEOUT et WRITE_TIMEOUT est spécifié dans le Tableau 179.

Tableau 179 – Attributs de READ/WRITE_TIMEOUT

Usage	Attribut	Description
m	integer	Temps en ms pendant lequel l'application EDD attend qu'un dispositif termine sa mise à jour demandée.

7.33.2.13 REFRESH_ACTIONS

Objet

L'attribut REFRESH_ACTIONS (actions de réactualisation) spécifie les METHOD qui doivent être exécutées lorsque la variable est affichée ou réactualisée. Les METHOD spécifiées doivent être exécutées dans l'ordre dans lequel elles apparaissent. Si une METHOD se termine pour une raison imprévue, les METHOD suivantes ne sont pas exécutées.

L'attribut REFRESH_ACTIONS ne doit pas être exprimé en utilisant une instruction conditionnelle (IF, IF-ELSE, SELECT). Si un comportement conditionnel est requis, les méthodes appelées par l'action doivent être compatibles avec celui-ci.

Structure lexicale

```
REFRESH_ACTIONS [reference] +, DEFINITION
```

Les attributs de REFRESH_ACTIONS sont spécifiés dans le Tableau 178.

7.33.2.14 STYLE**Objet**

L'attribut STYLE spécifie le mode d'affichage d'une VARIABLE. C'est une information additionnelle qui est individuellement interprétée par l'application EDDL.

Structure lexicale

```
STYLE (string) <exp>
```

L'attribut de STYLE est spécifié dans le Tableau 180.

Tableau 180 – Attribut de STYLE

Usage	Attribut	Description
m	string	Chaîne utilisée pour l'affichage.

7.34 VARIABLE_LIST**Objet**

Une VARIABLE_LIST (liste de variables) est un groupe d'objets de communication d'EDD (VARIABLE, VALUE_ARRAY ou RECORDS). Les VARIABLE_LISTS sont utilisés pour grouper des objets afin de faciliter l'exécution des applications.

NOTE Une VARIABLE_LIST peut être transférée en tant qu'objet de communication.

Structure lexicale

```
VARIABLE_LIST identifieur, [MEMBERS, HELP, LABEL, RESPONSE_CODES]
```

Les attributs sont spécifiés dans le Tableau 181.

Tableau 181 – Attributs de VARIABLE_LIST

Usage	Attribut	Description
m	MEMBERS	Élément lexical (voir 7.36.12).
o	HELP	Élément lexical (voir 7.36.8).
o	LABEL	Élément lexical (voir 7.36.9).
o	RESPONSE_CODES	Élément lexical (voir 7.36.14).

7.35 WAVEFORM**7.35.1 Structure générale****Objet**

Une WAVEFORM (forme d'onde) décrit un ensemble de données qui peut être affiché par un GRAPH (graphique).

Structure lexicale

```
WAVEFORM identifieur [TYPE, EMPHASIS, EXIT_ACTIONS, HANDLING, HELP, INIT_ACTIONS, KEY_POINTS, LABEL, LINE_COLOR, LINE_TYPE, REFRESH_ACTIONS, VALIDITY, Y_AXIS]
```

Les attributs de WAVEFORM sont spécifiés dans le Tableau 182.

Tableau 182 – Attributs de WAVEFORM

Usage	Attribut	Description
m	TYPE	Élément lexical (voir 7.35.2.1).
o	EMPHASIS	Élément lexical (voir 7.36.5).
o	EXIT_ACTIONS	Élément lexical (voir 7.35.2.2).
o	HANDLING	Élément lexical (voir 7.36.6).
o	HELP	Élément lexical (voir 7.36.8).
o	INIT_ACTIONS	Élément lexical (voir 7.35.2.3).
o	KEY_POINTS	Élément lexical (voir 7.35.2.4).
o	LABEL	Élément lexical (voir 7.36.9).
o	LINE_COLOR	Élément lexical (voir 7.36.10).
o	LINE_TYPE	Élément lexical (voir 7.36.11).
o	REFRESH_ACTIONS	Élément lexical (voir 7.35.2.5).
o	VALIDITY	Élément lexical (voir 7.36.16).
o	Y_AXIS	Élément lexical (voir 7.35.2.6).

7.35.2 Attributs spécifiques

7.35.2.1 TYPE

7.35.2.1.1 Structure générale

Objet

L'attribut TYPE spécifie le type de données contenues dans la WAVEFORM.

Structure lexicale

TYPE [XY, YT, HORIZONTAL, VERTICAL]

Les attributs de TYPE sont spécifiés dans le Tableau 183.

Tableau 183 – Attribut de TYPE

Usage	Attribut	Description
s	XY	Élément lexical (voir 7.35.2.1.2.1).
s	YT	Élément lexical (voir 7.35.2.1.2.2).
s	HORIZONTAL	Élément lexical (voir 7.35.2.1.2.3).
s	VERTICAL	Élément lexical (voir 7.35.2.1.2.4).

7.35.2.1.2 Attributs spécifiques

7.35.2.1.2.1 XY

Objet

L'attribut XY spécifie une WAVEFORM qui contient une liste de points (x,y).

Structure lexicale

XY [X_VALUES, Y_VALUES, NUMBER_OF_POINTS]

Les attributs de XY sont spécifiés dans le Tableau 184.

Tableau 184 – Attribut XY

Usage	Attribut	Description
m	X_VALUES	Objet L'attribut X_VALUES (valeurs X) spécifie la coordonnée X de chaque point dans la WAVEFORM. Chaque coordonnée X spécifiée dans X_VALUES doit correspondre à une coordonnée Y spécifiée dans Y_VALUES. Structure lexicale X_VALUES [reference, double, float, integer, unsigned_integer]+ Les attributs de X_VALUES sont spécifiés dans le Tableau 190.
m	Y_VALUES	Objet L'attribut Y_VALUES (valeurs Y) spécifie la coordonnée Y de chaque point dans la WAVEFORM. Chaque coordonnée Y spécifiée dans Y_VALUES doit correspondre à une coordonnée X spécifiée dans X_VALUES. Structure lexicale Y_VALUES [(reference, double, float, integer, unsigned_integer)+ Les attributs de Y_VALUES sont spécifiés dans le Tableau 190.
o	NUMBER_OF_POINTS	Objet L'attribut NUMBER_OF_POINTS spécifie le nombre de points valides dans X_VALUES et Y_VALUES. Par défaut, le nombre de points dans une WAVEFORM sans attribut NUMBER_OF_POINTS est égal à la taille de X_VALUES et Y_VALUES. Structure lexicale NUMBER_OF_POINTS (integer)<exp>

7.35.2.1.2.2 YT**Objet**

L'attribut YT spécifie une WAVEFORM qui contient une liste de points qui sont définis par une coordonnée X initiale, un incrément X entre des points successifs et une liste de valeurs Y.

Structure lexicale

YT [X_INITIAL, X_INCREMENT, Y_VALUES, NUMBER_OF_POINTS]

Les attributs de YT sont spécifiés dans le Tableau 185.

Tableau 185 – Attributs de YT

Usage	Attribut	Description
m	X_INITIAL	Objet L'attribut X_INITIAL (X initial) spécifie la coordonnée X du premier point dans la WAVEFORM. Structure lexicale X_INITIAL (reference)<exp> X_INITIAL (double)<exp> X_INITIAL (float)<exp> X_INITIAL (integer)<exp> X_INITIAL (unsigned_integer)<exp> La référence est une référence à une instance de VARIABLE, un membre d'une instance de RECORD, un élément d'une instance de VALUE_ARRAY ou un élément d'une instance de REFERENCE_ARRAY.
m	X_INCREMENT	Objet L'attribut X_INCREMENT (incrément X) spécifie la différence entre les coordonnées X de points adjacents dans la WAVEFORM. Structure lexicale X_INCREMENT (reference)<exp> X_INCREMENT (double)<exp> X_INCREMENT (float)<exp> X_INCREMENT (integer)<exp>

Usage	Attribut	Description
		X_INCREMENT (unsigned_integer)<exp> La référence est une référence à une instance de VARIABLE, un membre d'une instance de RECORD, un élément d'une instance de VALUE_ARRAY ou un élément d'une instance de REFERENCE_ARRAY.
m	Y_VALUES	Objet L'attribut Y_VALUES (valeurs Y) spécifie la coordonnée Y de chaque point dans la WAVEFORM. Structure lexicale Y_VALUES [(reference, double, float, integer, unsigned_integer)<exp>]+ Les attributs de Y_VALUES sont spécifiés dans le Tableau 190.
o	NUMBER_OF_POINTS	Objet L'attribut NUMBER_OF_POINTS spécifie le nombre de points valides dans X_VALUES et Y_VALUES. Par défaut, le nombre de points dans une WAVEFORM sans attribut NUMBER_OF_POINTS est égal à la taille de X_VALUES et Y_VALUES Structure lexicale NUMBER_OF_POINTS (integer)<exp>

7.35.2.1.2.3 HORIZONTAL

Objet

L'attribut HORIZONTAL spécifie une WAVEFORM qui contient des lignes horizontales.

Structure lexicale

HORIZONTAL Y_VALUES

L'attribut d'HORIZONTAL est spécifié dans le Tableau 186.

Tableau 186 – Attribut d'HORIZONTAL

Usage	Attribut	Description
m	Y_VALUES	Objet L'attribut Y_VALUES spécifie la coordonnée Y de chaque ligne horizontale dans la WAVEFORM. Structure lexicale Y_VALUES [reference, double, float, integer, unsigned_integer]+ Les attributs de Y_VALUES sont spécifiés dans le Tableau 190.

7.35.2.1.2.4 VERTICAL

Objet

L'attribut VERTICAL spécifie une WAVEFORM qui contient des lignes verticales.

Structure lexicale

VERTICAL X_VALUES

Les attributs de VERTICAL sont spécifiés dans le Tableau 187.

Tableau 187 – Attribut de VERTICAL

Usage	Attribut	Description
m	X_VALUES	Objet L'attribut X_VALUES spécifie la coordonnée X de chaque ligne verticale dans la WAVEFORM. Structure lexicale X_VALUES [reference, double, float, integer, unsigned_integer]+ Les attributs de X_VALUES sont spécifiés dans le Tableau 190.

7.35.2.2 EXIT_ACTIONS

Objet

L'attribut EXIT_ACTIONS (actions de sortie) spécifie les actions qui sont exécutées lorsque le GRAPH contenant la WAVEFORM est fermé.

Les METHOD spécifiées doivent être exécutées dans l'ordre dans lequel elles apparaissent. Si une méthode se termine pour une raison imprévue, les METHOD restantes ne doivent pas être exécutées.

L'attribut EXIT_ACTIONS ne doit pas être exprimé en utilisant une instruction conditionnelle (IF, IF-ELSE, SELECT). Si un comportement conditionnel est requis, les méthodes appelées par l'action doivent être compatibles avec celui-ci.

Structure lexicale

EXIT_ACTIONS [reference]+, DEFINITION

Les attributs d'EXIT_ACTIONS sont spécifiés dans le Tableau 188.

7.35.2.3 INIT_ACTIONS

Objet

L'attribut INIT_ACTIONS (actions d'initiation) spécifie les actions qui sont exécutées avant que la WAVEFORM soit initialement affichée par un GRAPH. Ces méthodes sont généralement utilisées pour charger la WAVEFORM avec les données provenant d'un dispositif ou d'un FILE (fichier).

Les METHOD spécifiées doivent être exécutées dans l'ordre dans lequel elles apparaissent. Si une méthode se termine pour une raison imprévue, les METHOD restantes ne doivent pas être exécutées. De plus, l'application EDD ne doit pas dessiner la WAVEFORM si une METHOD se termine pour une raison imprévue.

L'attribut INIT_ACTIONS ne doit pas être exprimé en utilisant une instruction conditionnelle (IF, IF-ELSE, SELECT). Si un comportement conditionnel est requis, les méthodes appelées par l'action doivent être compatibles avec celui-ci.

Structure lexicale

INIT_ACTIONS [reference]+, DEFINITION

Les attributs d'INIT_ACTIONS sont spécifiés dans le Tableau 188.

Tableau 188 – Attributs d'EXIT_ACTIONS, INIT_ACTIONS, REFRESH_ACTIONS

Usage	Attribut	Description
o	reference	Référence à une instance de METHOD ou une instance de METHOD avec arguments.
o	DEFINITION	Élément lexical (voir 7.36.4).

7.35.2.4 KEY_POINTS**Structure générale****Objet**

L'attribut KEY_POINTS (points clés) spécifie les points clés dans la WAVEFORM qu'il convient de mettre en évidence à l'aide de l'application EDD. Les points clés ne correspondent pas nécessairement directement aux points de données spécifiés via l'attribut TYPE. La façon selon laquelle ces points sont mis en surveillance est définie par la spécification de l'EDD. Par défaut, il convient qu'une WAVEFORM sans attribut KEY_POINTS ne comporte aucun point mis en évidence.

Structure lexicale

KEY_POINTS [X_VALUES, Y_VALUES]

Les attributs de KEY_POINTS sont spécifiés dans le Tableau 189.

Tableau 189 – Attributs de KEY_POINTS

Usage	Attribut	Description
m	X_VALUES	Coordonnée X de chaque KEY_POINT.
m	Y_VALUES	Coordonnée Y de chaque KEY_POINT.

7.35.2.4.2 Attributs spécifiques**7.35.2.4.2.1 X_VALUES****Objet**

L'attribut X_VALUES (valeurs X) spécifie la coordonnée X de chaque point qui doit être mis en évidence. Chaque coordonnée X spécifiée dans X_VALUES doit correspondre à une coordonnée Y spécifiée dans Y_VALUES.

Structure lexicale

X_VALUES [reference, double, float, integer, unsigned_integer]+

Les attributs de X_VALUES sont spécifiés dans le Tableau 190.

Tableau 190 – Attribut de X_VALUES, Y_VALUES

Usage	Attribut	Description
o	reference	Référence à une instance de VARIABLE, un membre d'une instance de RECORD, un élément d'une instance de VALUE_ARRAY, un élément d'une instance de REFERENCE_ARRAY, une instance de VALUE_ARRAY ou une instance de REFERENCE_ARRAY.
o	double	Constante à virgule flottante à double précision.
o	float	Constante à virgule flottante à simple précision.
o	integer	Constante entière.
o	unsigned_integer	Constante entière non signée.

7.35.2.4.2.2 Y_VALUES

Objet

L'attribut Y_VALUES spécifie la coordonnée Y de chaque point à mettre en évidence. Chaque coordonnée Y spécifiée dans Y_VALUES doit correspondre à une coordonnée X spécifiée dans X_VALUES.

Structure lexicale

Y_VALUES [reference, double, float, integer, unsigned_integer]+

Les attributs de Y_VALUES sont spécifiés dans le Tableau 190.

7.35.2.5 REFRESH_ACTIONS

Objet

L'attribut REFRESH_ACTIONS (actions de réactualisation) spécifie les actions qui sont exécutées chaque fois que des modifications sont apportées au X_AXIS ou Y_AXIS du GRAPH dans lequel la WAVEFORM est affichée.

Les METHOD spécifiées doivent être exécutées dans l'ordre dans lequel elles apparaissent. Si une METHOD se termine pour une raison imprévue, les METHOD restantes ne doivent pas être exécutées. De plus, les X_VALUES et Y_VALUES de la WAVEFORM doivent être restaurées à leurs valeurs originales par l'application EDD, et la WAVEFORM sur le GRAPH ne doit pas être redessinée.

L'attribut REFRESH_ACTIONS ne doit pas être exprimé en utilisant une instruction conditionnelle (IF, IF-ELSE, SELECT). Si un comportement conditionnel est requis, les méthodes appelées par l'action doivent être compatibles avec celui-ci.

Structure lexicale

REFRESH_ACTIONS [reference]+, DEFINITION

Les attributs de REFRESH_ACTIONS sont spécifiés dans le Tableau 188.

7.35.2.6 Y_AXIS

Objet

L'attribut Y_AXIS spécifie le mode d'affichage de l'axe Y de la WAVEFORM. Chaque WAVEFORM sur un GRAPH partage un X_AXIS commun mais chaque WAVEFORM sur un GRAPH a son propre Y_AXIS. Par défaut, l'affichage de l'axe Y d'un GRAPH sans attribut Y_AXIS est dérivé des données dans la WAVEFORM.

Structure lexicale

Y_AXIS (reference) <exp>

L'attribut de Y_AXIS est spécifié dans le Tableau 191.

Tableau 191 – Attribut de Y_AXIS

Usage	Attribut	Description
m	reference	Référence à une instance d'AXIS.

7.36 Attributs communs

7.36.1 CLASSIFICATION

Objet

L'attribut CLASSIFICATION est une valeur énumérée unique qui indique les niveaux (éventuellement multiples) de classification pour ce dispositif.

Structure lexicale

CLASSIFICATION [voir les attributs dans le Tableau 192]

Les attributs de CLASSIFICATION sont spécifiés dans le Tableau 192.

Tableau 192 – Attribut de CLASSIFICATION

Usage	Attribut	Description
s	ACTUATOR	Le composant représente un actionneur.
s	ACTUATOR_ELECTRO_PNEUMATIC	Le composant représente un actionneur électropneumatique.
s	ACTUATOR_ELECTRIC	Le composant représente un actionneur électrique.
s	ACTUATOR_HYDRAULIC	Le composant représente un actionneur hydraulique.
s	CONVERTER	Le composant représente un convertisseur.
s	CONTROLLER	Le composant représente un contrôleur.
s	DISCRETE_IN	Le composant représente une entrée discrète.
s	DISCRETE_OUT	Le composant représente une sortie discrète.
s	ELECTRICAL_DISTRIBUTION	Le composant représente une distribution électrique.
s	ELECTRICAL_DISTRIBUTION_POWER_MONITORING	Le composant représente un dispositif de surveillance de puissance de distribution électrique.
s	FREQUENCY_CONVERTER	Le composant représente un convertisseur de fréquence.
s	INDICATOR	Le composant représente un indicateur.
s	NETWORK_COMPONENT	Le composant représente un réseau.
s	NETWORK_COMPONENT_WIRELESSADAPTER	Le composant représente une carte réseau sans fil.
s	REMOTEIO	Le composant représente une E/S distante.
s	RECORDER	Le composant représente un enregistreur.
s	SENSOR	Le composant représente un capteur.
s	SENSOR_ACOUSTIC	Le composant représente un capteur acoustique.
s	SENSOR_ANALYTIC	Le composant représente un capteur analytique.
s	SENSOR_ANALYTIC_GAS	Le composant représente un capteur analytique de gaz.
s	SENSOR_ANALYTIC_LIQUID	Le composant représente un capteur analytique de liquide.

Usage	Attribut	Description
s	SENSOR_CONCENTRATION	Le composant représente un capteur de concentration.
s	SENSOR_DENSITY	Le composant représente un capteur de densité.
s	SENSOR_DENSITY_RADIOMETRIC	Le composant représente un capteur de densité radiométrique.
s	SENSOR_FLOW	Le composant représente un capteur de flux.
s	SENSOR_FLOW_CORIOLIS	Le composant représente un capteur de flux Coriolis.
s	SENSOR_FLOW_ELECTRO_MAGNETIC	Le composant représente un capteur de flux électromagnétique.
s	SENSOR_FLOW_MECHANICAL	Le composant représente un capteur de flux mécanique.
s	SENSOR_FLOW_RADIOMETRIC	Le composant représente un capteur de flux radiométrique.
s	SENSOR_FLOW_THERMAL	Le composant représente un capteur de flux thermique.
s	SENSOR_FLOW_ULTRASONIC	Le composant représente un capteur de flux ultrasonore.
s	SENSOR_FLOW_VORTEX_COUNTER	Le composant représente un capteur de flux à compteur vortex.
s	SENSOR_LEVEL	Le composant représente un capteur de niveau.
s	SENSOR_LEVEL_BUOYANCY	Le composant représente un capteur de niveau de flottaison.
s	SENSOR_LEVEL_CAPACITIVE	Le composant représente un capteur de niveau capacitif.
s	SENSOR_LEVEL_ECHO	Le composant représente un capteur de niveau à écho.
s	SENSOR_LEVEL_HYDROSTATIC	Le composant représente un capteur de niveau hydrostatique.
s	SENSOR_LEVEL_RADIOMETRIC	Le composant représente un capteur de niveau radiométrique.
s	SENSOR_MULTIVARIABLE	Le composant représente un capteur multivariable.
s	SENSOR_POSITION	Le composant représente un capteur de position.
s	SENSOR_PRESSURE	Le composant représente un capteur de pression.
s	SENSOR_TEMPERATURE	Le composant représente un capteur de température.
s	SEMICONDUCTOR	Le composant représente un semi-conducteur.
s	SWITCHGEAR	Le composant représente un dispositif de commutation.
s	UNIVERSAL	Le composant représente un composant universel.
s	string	Le composant représente la chaîne.

7.36.2 COMPONENT_PARENT

Objet

L'attribut COMPONENT_PARENT (composant parent) est une référence à un COMPONENT_FOLDER (dossier de composant). COMPONENT_PARENT peut être utilisé à

la place de **PROTOCOL** et **CLASSIFICATION** pour simplifier (raccourcir) la déclaration de **COMPONENT_PATH**. Si le **PROTOCOL** et/ou la **CLASSIFICATION** est défini dans le **COMPONENT_PARENT** (ou son parent), le **PROTOCOL** et/ou la **CLASSIFICATION** ne doivent pas être redéfinis. Si le **COMPONENT_PARENT** est spécifié, le **COMPONENT_PATH** (chemin de composant) doit être défini.

NOTE Le **COMPONENT_PARENT** est prédéfini par la structure de bibliothèque d'EDD HCF et FF.

Structure lexicale

COMPONENT_PARENT string

L'attribut de **COMPONENT_PARENT** est spécifié dans le Tableau 193.

Tableau 193 – Attribut de COMPONENT_PARENT

Usage	Attribut	Description
m	string	Composant parent dans le catalogue

7.36.3 COMPONENT_PATH

Objet

L'attribut **COMPONENT_PATH** (chemin du composant) permet au fabricant d'EDD de désigner l'emplacement dans le catalogue d'une application EDD. Le chemin spécifie uniquement la partie spécifique au fabricant de l'emplacement sans le **PROTOCOL**, la **CLASSIFICATION** et le **MANUFACTURER**. Si un **COMPONENT_PARENT** est utilisé, le **PROTOCOL**, la **CLASSIFICATION** et le **MANUFACTURER** ne peuvent pas être définis et le **COMPONENT_PATH** définit uniquement la sous-hiérarchie.

NOTE Le **COMPONENT_PATH** est prédéfini par la structure de bibliothèque d'EDD HCF et FF.

Structure lexicale

COMPONENT_PATH string

Les attributs de **COMPONENT_PATH** sont spécifiés dans le Tableau 194.

Tableau 194 – Attribut de COMPONENT_PATH

Usage	Attribut	Description
m	string	Chemin d'accès spécifique au fabricant du composant dans le catalogue.

7.36.4 DEFINITION

Objet

L'attribut **DEFINITION** (définition) spécifie les actions à effectuer et est basé sur un langage de programmation procédural (par exemple, ANSI C). Dans les **DEFINITION**, des **METHOD** et des **Builtin** (voir Annexe B) peuvent être appelés. Des éléments de base de l'EDD peuvent être utilisés dans une **DEFINITION**.

Les éléments suivants doivent être pris en charge.

- Types de données de base (voir D.5.1).
- Matrices
- Opérateurs arithmétiques
- Instructions
- Invocation de **METHOD** et de **Builtin**

- METHOD avec arguments (appel par valeur, appel par référence)

Un appel par valeur est utilisé pour transmettre une copie des données à la METHOD. Des changements de la copie dans la METHOD ne modifient pas les données originales.

L'appel par référence est utilisé pour accéder directement aux données. Les changements affectent les données originales.

Structure lexicale

DEFINITION compound-statement

L'attribut de DEFINITION est spécifié dans le Tableau 195.

Tableau 195 – Attribut de DEFINITION

Usage	Attribut	Description
o	compound-statement	Bloc d'instructions pour un langage de programmation procédural pris en charge.

7.36.5 EMPHASIS

Objet

L'attribut EMPHASIS (mise en évidence) spécifie s'il convient qu'un élément d'interface utilisateur soit mis en évidence lorsqu'il est affiché. La façon selon laquelle il est souligné est définie par l'application EDD. Par défaut, il convient qu'un élément d'interface utilisateur sans attribut EMPHASIS ne soit pas mis en évidence.

Structure lexicale

EMPHASIS (FALSE, TRUE) <exp>

Les attributs d'EMPHASIS sont spécifiés dans le Tableau 196.

Tableau 196 – Attribut d'EMPHASIS

Usage	Attribut	Description
s	FALSE	Il convient que l'élément d'interface utilisateur ne soit pas souligné.
s	TRUE	Il convient que l'élément d'interface utilisateur soit souligné.

7.36.6 HANDLING

Objet

L'attribut HANDLING (manipulation) spécifie les opérations qui peuvent être effectuées sur un élément. Par défaut, un élément sans attribut HANDLING peut être lu et écrit.

Structure lexicale

HANDLING (READ, READ_WRITE, WRITE) <exp>

Les attributs de HANDLING sont spécifiés dans le Tableau 197.

Tableau 197 – Attributs de HANDLING

Usage	Attribut	Description
s	READ	L'élément peut être uniquement lu.
s	READ_WRITE	L'élément peut être lu et écrit.
s	WRITE	L'élément peut être écrit.

7.36.7 HEIGHT

Objet

L'attribut HEIGHT (hauteur) spécifie la taille verticale d'un élément d'interface utilisateur lorsqu'il est affiché. Par défaut, la hauteur d'un élément d'interface utilisateur sans attribut HEIGHT est MEDIUM (moyenne).

Structure lexicale

HEIGHT (XX_SMALL, X_SMALL, SMALL, MEDIUM, LARGE, X_LARGE, XX_LARGE)

Les attributs de HEIGHT sont spécifiés dans le Tableau 198.

Tableau 198 – Attributs de HEIGHT/WIDTH

Usage	Attribut	Description
s	XX_SMALL	Il convient que l'élément d'interface utilisateur occupe une partie extrêmement petite de la fenêtre dans laquelle il est affiché.
s	X_SMALL	Il convient que l'élément d'interface utilisateur occupe une très petite partie de la fenêtre dans laquelle il est affiché.
s	SMALL	Il convient que l'élément d'interface utilisateur occupe une petite partie de la fenêtre dans laquelle il est affiché.
s	MEDIUM	Il convient que l'élément d'interface utilisateur occupe une partie moyenne de la fenêtre dans laquelle il est affiché.
s	LARGE	Il convient que l'élément d'interface utilisateur occupe une grande partie de la fenêtre dans laquelle il est affiché.
s	X_LARGE	Il convient que l'élément d'interface utilisateur occupe une très grande partie de la fenêtre dans laquelle il est affiché.
s	XX_LARGE	Il convient que l'élément d'interface utilisateur occupe une partie extrêmement grande de la fenêtre dans laquelle il est affiché.

7.36.8 HELP

Objet

L'attribut HELP (aide) spécifie du texte, qui donne une description de la construction.

NOTE Ce texte est destiné à être utilisé par des applications EDD pour information des utilisateurs.

Structure lexicale

HELP (string)<exp>

L'attribut de HELP est spécifié dans le Tableau 199.

Tableau 199 – Attribut de HELP

Usage	Attribut	Description
m	string	Texte lisible par l'utilisateur.

7.36.9 LABEL

Objet

L'attribut LABEL (étiquette) spécifie la désignation affichée d'un élément.

Structure lexicale

LABEL (string)<exp>

L'attribut de LABEL est spécifié dans le Tableau 200.

Tableau 200 – Attribut de LABEL

Usage	Attribut	Description
m	string	Texte lisible par l'utilisateur.

7.36.10 LINE_COLOR**Objet**

L'attribut LINE_COLOR (couleur de trait) spécifie la couleur qu'il convient d'utiliser pour afficher un élément d'interface utilisateur. Par défaut, un élément d'interface utilisateur sans attribut LINE_COLOR sera affiché en utilisant une couleur par défaut définie par l'application EDD.

Structure lexicale

LINE_COLOR (integer, reference)<exp>

Les couleurs sont spécifiées par l'intermédiaire d'une valeur RVB, comme défini dans Cascading Style Sheets, niveau 2.

Les attributs de LINE_COLOR sont spécifiés dans le Tableau 201.

Tableau 201 – Attributs de LINE_COLOR

Usage	Attribut	Description
s	integer	Couleur à utiliser pour afficher l'élément d'interface utilisateur.
s	reference	Référence à une instance de VARIABLE contenant la couleur à utiliser pour afficher l'élément d'interface utilisateur.

7.36.11 LINE_TYPE**Objet**

L'attribut LINE_TYPE (type de trait) spécifie le type de données contenues dans un élément d'interface utilisateur. Il convient que chaque élément d'interface utilisateur avec le même attribut LINE_TYPE soit affiché de la même façon (c'est-à-dire, couleur de trait, motif de trait, épaisseur de trait, etc.). Par défaut, un élément d'interface utilisateur sans attribut LINE_TYPE utilise le type de trait DATA.

Structure lexicale

LINE_TYPE ([DATA, n]*, LOW_LOW_LIMIT, LOW_LIMIT, HIGH_LIMIT, HIGH_HIGH_LIMIT, TRANSPARENT)<exp>

Les attributs de LINE_TYPE sont spécifiés dans le Tableau 202.

Tableau 202 – Attribut de LINE_TYPE

Usage	Attribut	Description
s	DATA	L'élément d'interface utilisateur contient des données.
c	n	Entier croissant commençant à 1, qui identifie des types de trait DATA distincts si plusieurs sont présents.
s	LOW_LOW_LIMIT	L'élément d'interface utilisateur contient une limite basse inférieure.
s	LOW_LIMIT	L'élément d'interface utilisateur contient une limite basse.
s	HIGH_LIMIT	L'élément d'interface utilisateur contient une limite haute.
s	HIGH_HIGH_LIMIT	L'élément d'interface utilisateur contient une limite haute supérieure.
s	TRANSPARENT	Seuls les points de données d'élément d'interface utilisateur sont

Usage	Attribut	Description
		visibles.

7.36.12 MEMBERS

Objet

L'attribut MEMBERS définit les membres d'une construction de base. Chaque membre spécifie un élément dans le groupe, et est défini par un ensemble de quatre attributs (identifiant, référence, description, aide).

L'attribut MEMBERS d'un FILE (fichier) ne doit pas être exprimé en utilisant une instruction conditionnelle (IF, IF-ELSE, SELECT).

Structure lexicale

MEMBERS [(identifiant, reference, description, help)<exp>]+

Les attributs de MEMBERS sont spécifiés dans le Tableau 203.

Tableau 203 – Attributs de MEMBERS

Usage	Attribut	Description
m	identifiant	Nom par lequel l'élément peut être référencé. Un nom d'identifiant spécifique peut être réutilisé dans l'attribut MEMBERS d'éléments multiples dans la mesure où les éléments sont du même type.
m	reference	Référence à un élément EDD. Le type de la référence dépend des constructions de base. Si MEMBERS est un attribut de CHART (diagramme), la référence est une référence à une instance de SOURCE. Si MEMBERS est un attribut de COLLECTION, la référence est une référence à une instance d'AXIS, BLOCK_A, BLOCK_B, CHART, COLLECTION, COMMAND, CONNECTION, DOMAIN, EDIT_DISPLAY, FILE, GRAPH, LIST, MENU, METHOD, PROGRAM, RECORD, REFERENCE_ARRAY, REFRESH, RESPONSE_CODES, SOURCE, UNIT, VALUE_ARRAY, VARIABLE, VARIABLE_LIST, WAVEFORM, ou WRITE_AS_ONE. Si MEMBERS est un attribut de FILE ou LIST, la référence est une référence à une instance de COLLECTION, LIST, RECORD, REFERENCE_ARRAY, VALUE_ARRAY ou VARIABLE. Pour une COLLECTION, le type de la COLLECTION doit être LIST, RECORD, REFERENCE_ARRAY, VALUE_ARRAY, ou VARIABLE. Si MEMBERS est un attribut de GRAPH, la référence est une référence à une instance de WAVEFORM. Si MEMBERS est un attribut de RECORDS, la référence est une référence à une instance de VARIABLE. Si MEMBERS est un attribut de SOURCE, la référence est une référence à une instance de VARIABLE ou un membre d'une instance de RECORD. Si MEMBERS est un attribut de VARIABLE_LISTS, la référence est une référence à une instance de RECORDS, ou VALUE_ARRAY ou VARIABLE.
o	description	Brève description de l'élément.
o	help	Texte d'aide pour l'élément.

7.36.13 PROTOCOL

Objet

L'attribut PROTOCOL (protocole) indique le protocole de communication utilisé pour échanger avec le dispositif modélisé par l'EDD en question. Le PROTOCOL est utilisé pour séparer les EDD dans le catalogue.

Structure lexicale

PROTOCOL [PROFIBUS_DP, PROFIBUS_PA, PROFINET, FF, HART, string]

Les attributs de PROTOCOL sont spécifiés dans le Tableau 204.

Tableau 204 – Attribut de PROTOCOL

Usage	Attribut	Description
s	FF	Le composant est un composant Fieldbus Foundation.
s	HART	Le composant est un composant HART.
s	PROFIBUS_DP	Le composant est un composant PROFIBUS DP.
s	PROFIBUS_PA	Le composant est un composant PROFIBUS PA.
s	PROFINET	Le composant est un composant PROFINET.
s	string	Spécifie un protocole quelconque. Le protocole doit être pris en charge par l'application EDD utilisée.

7.36.14 RESPONSE_CODES

Objet

L'attribut RESPONSE_CODES (codes de réponse) spécifie les valeurs qu'un dispositif peut retourner en tant qu'information d'erreur. Chaque VARIABLE, RECORD, VALUE_ARRAY, VARIABLE_LIST, PROGRAM, COMMAND ou DOMAIN peut avoir son propre ensemble de RESPONSE_CODES.

Deux structures lexicales pour l'attribut RESPONSE_CODES sont fournies:

- le RESPONSE_CODES référencé contient une référence à une instance de RESPONSE_CODES;
- le RESPONSE_CODES incorporé permet de spécifier les éléments RESPONSE_CODES directement à l'instance d'élément.

Structure lexicale pour les RESPONSE_CODES référencés

RESPONSE_CODES (reference) <exp>

L'attribut du RESPONSE_CODES référencé est spécifié dans le Tableau 205.

Tableau 205 – Attribut de RESPONSE_CODES

Usage	Attribut	Description
m	reference	Référence à une instance de RESPONSE_CODES.

Structure lexicale pour les RESPONSE_CODES incorporés

RESPONSE_CODES [(description, integer, [DATA_ENTRY_ERROR, DATA_ENTRY_WARNING, MISC_ERROR, MISC_WARNING, MODE_ERROR, PROCESS_ERROR, SUCCESS], help) <exp>]+

Les attributs de RESPONSE_CODES incorporés sont spécifiés dans le Tableau 154.

7.36.15 SUPPLIED_INTERFACE

Objet

L'attribut SUPPLIED_INTERFACE (interface fournie) répertorie toutes les interfaces prises en charge par ce composant.

Structure lexicale

SUPPLIED_INTERFACE [interface-reference] +

L'attribut de SUPPLIED_INTERFACE est spécifié dans le Tableau 206.

Tableau 206 – Attribut de SUPPLIED_INTERFACE

Usage	Attribut	Description
m	interface-reference	Liste d'identifiants d'interface.

7.36.16 VALIDITY

Objet

L'attribut VALIDITY (validité) spécifie si un élément est ou n'est pas valide. Un élément sans attribut VALIDITY est toujours valide.

NOTE 1 Il convient que VALIDITY soit exprimé en utilisant une instruction conditionnelle (IF, IF-ELSE, SELECT).

NOTE 2 Si un élément d'interface utilisateur n'est pas valide, il convient qu'il ne soit pas affiché.

NOTE 3 Une METHOD doit être interrompue lorsqu'elle accède à une VARIABLE non valide.

Structure lexicale

VALIDITY [(FALSE, TRUE) <exp>]

Les attributs de VALIDITY sont spécifiés dans le Tableau 207.

Tableau 207 – Attributs de VALIDITY

Usage	Attribut	Description
s	FALSE	L'élément n'est pas valide.
s	TRUE	L'élément est valide.

7.36.17 WIDTH

Objet

L'attribut WIDTH (largeur) spécifie la taille horizontale d'un élément d'interface utilisateur lorsqu'il est affiché. Par défaut, la largeur d'un élément d'interface utilisateur sans attribut WIDTH est MEDIUM (moyenne).

Structure lexicale

WIDTH (XX_SMALL, X_SMALL, SMALL, MEDIUM, LARGE, X_LARGE, XX_LARGE)

Les attributs de WIDTH sont spécifiés dans le Tableau 198.

7.37 Expression conditionnelle

Objet

Des expressions conditionnelles sont utilisées pour déclarer des valeurs alternatives ou calculer des valeurs pour un attribut d'un élément de base. La valeur de l'expression est évaluée pendant le temps d'exécution. L'instruction conditionnelle IF ou SELECT peut être utilisée pour activer ou désactiver des instances référencées d'une liste d'attributs ITEMS de MENU, etc.

Trois types d'expressions conditionnelles peuvent être utilisés.

- Une instruction conditionnelle IF est utilisée pour spécifier un attribut qui a deux définitions alternatives. L'expression spécifiée dans l'instruction conditionnelle IF est évaluée. Si le résultat est différent de zéro, l'attribut est spécifié par une clause then-clause ("alors"); sinon, il est spécifié par une clause else-clause ("sinon").
- Une instruction conditionnelle SELECT est utilisée pour spécifier un attribut qui a plusieurs définitions alternatives. L'expression select-expression est évaluée et comparée à chaque integer_n de CASE (cas). Si une correspondance est trouvée, la select-clause appropriée est valide. Si aucun integer_n correspondant n'est trouvé, la clause par défaut default-clause suivant DEFAULT est utilisée.
- Une expression primaire, unaire ou binaire (voir 7.39.7). Ce type d'expression conditionnelle peut être utilisé uniquement pour des valeurs numériques (par exemple, DEFAULT_VALUE, MIN_VALUE, SLOT, INDEX).

EXEMPLE Il est possible de définir plus d'une valeur pour une DEFAULT_VALUE (valeur par défaut), MIN_VALUE (valeur minimale), etc. en utilisant une instruction conditionnelle IF ou SELECT.

Structure lexicale pour l'instruction conditionnelle IF

IF (if-expression), (then-clause)+ ELSE (else-clause)+

Les attributs sont spécifiés dans le Tableau 208.

Structure lexicale pour l'instruction conditionnelle SELECT

SELECT select-expression, (CASE integer_n, select-clause)+, (DEFAULT, default-clause)

Les attributs sont spécifiés dans le Tableau 208.

Tableau 208 – Instruction conditionnelle IF, SELECT

Usage	Attribut	Description
m	if-expression	Évaluée pour déterminer si la clause then-clause ou else-clause est utilisée pour définir un attribut. Si l'expression if-expression contient une référence à une instance de VARIABLE, la valeur de la VARIABLE référencée est utilisée.
m	then-clause	Définition pour l'attribut si la valeur de condition est différente de zéro. La structure de clause dépend de l'attribut défini. Elle peut également prendre la forme d'une autre instruction conditionnelle.
o	else-clause	Définition pour l'attribut si la valeur de condition est zéro. La structure de clause dépend de l'attribut défini. Elle peut également prendre la forme d'une autre instruction conditionnelle.
m	select-expression	Évalué pour déterminer quelle clause select-clause est utilisée pour définir un attribut. Si l'expression select-expression contient une référence à une instance de VARIABLE, la valeur de la VARIABLE référencée est utilisée.
m	integer_n	Numéro identifiant le CASE correspondant. Si un entier correspond à integer_n, la select-clause correspondante est utilisée.
m	select-clause	Définition pour l'attribut pour chaque CASE, et définition DEFAULT (par défaut). La structure de chaque clause dépend de l'attribut défini. Indépendamment de l'attribut, chaque clause peut également prendre la forme d'une autre instruction conditionnelle.
o	default-clause	Définition pour la définition DEFAULT (par défaut). La structure de default-clause dépend de l'attribut défini. Indépendamment de l'attribut, chaque clause peut également prendre la forme d'une autre instruction conditionnelle.

Si la clause

- then-clause,
- else-clause,
- select-clause,

- default-clause

contient des références à des instances de METHOD ou des Builtin, la valeur retournée de la METHOD ou du Builtin référencé doit être utilisée dans la clause. Les valeurs de retour doivent être compatibles au niveau du type.

7.38 Référencement

7.38.1 Référencement d'une instance d'EDD

Objet

Des références sont utilisées dans une EDD pour faire référence à d'autres éléments de l'EDD.

EXEMPLE Un élément PRE_EDIT_ACTIONS (actions pré-édition) de VARIABLE fait référence à des instances de METHOD définies ailleurs dans l'EDD.

Structure lexicale

L'attribut est spécifié dans le Tableau 209.

Tableau 209 – Référencement d'une instance d'EDD

Usage	Attribut	Description
m	identifier	Identifiant d'une instance d'EDD

7.38.2 Référencement de bits d'une VARIABLE BIT ENUMERATED

Objet

Ce référencement est utilisé pour accéder à un bit d'une instance VARIABLE BIT ENUMERATED.

Structure lexicale

variable-reference, bit-mask

Les attributs sont spécifiés dans le Tableau 210.

Tableau 210 – Référencement d'éléments de VARIABLE

Usage	Attribut	Description
m	variable-reference	Référence à une instance de VARIABLE; le type de l'instance de VARIABLE doit être BIT ENUMERATED.
m	member-identifier	Masque binaire qui identifie un bit unique dans la variable; c'est le masque utilisé pour définir le bit dans la variable.

7.38.3 Référencement des membres d'un RECORD

Objet

Ce référencement est utilisé pour accéder à un membre d'une instance de RECORD.

Structure lexicale

record-reference, member-identifier

Les attributs sont spécifiés dans le Tableau 211.

Tableau 211 – Référencement d'éléments de RECORD

Usage	Attribut	Description
m	record-reference	Référence à une instance de RECORD.
m	member-identifier	Identifiant d'un élément de MEMBERS.

7.38.4 Référencement d'éléments d'un VALUE_ARRAY**Objet**

Ce référencement est utilisé pour accéder à un élément d'une instance de VALUE_ARRAY.

Structure lexicale

value-array-reference, expression

Les attributs sont spécifiés dans le Tableau 183.

Tableau 212 – Référencement d'éléments de VALUE_ARRAY

Usage	Attribut	Description
m	value-array-reference	Référence à une instance de VALUE_ARRAY.
m	expression	Expression qui doit être égale à un entier qui n'est pas négatif qui est dans l'intervalle d'index de l'attribut VALUE_ARRAY (pour une description des expressions voir 7.40).

7.38.5 Référencement des membres d'une COLLECTION**Objet**

Ce référencement est utilisé pour accéder à un membre d'une instance de COLLECTION.

Structure lexicale

collection-reference, member-identifier

Les attributs sont spécifiés dans le Tableau 213.

Tableau 213 – Référencement des membres de COLLECTION

Usage	Attribut	Description
m	collection-reference	Référence à une instance de COLLECTION.
m	member-identifier	Identifiant d'un élément de MEMBERS.

7.38.6 Référencement d'éléments d'un REFERENCE_ARRAY**Objet**

Ce référencement est utilisé pour accéder à un élément d'une instance de REFERENCE_ARRAY.

Structure lexicale

reference-array-reference, expression

Les attributs sont spécifiés dans le Tableau 214.

Tableau 214 – Référencement des membres de REFERENCE_ARRAY

Usage	Attribut	Description
m	reference-array-reference	Référence à une instance de REFERENCE_ARRAY.
m	expression	Expression qui doit être égale à un entier n'est pas négatif qui est dans l'intervalle d'index du REFERENCE_ARRAY (pour la description des expressions, voir 7.39.7).

7.38.7 Référencement des membres d'une VARIABLE_LISTS**Objet**

Ce référencement est utilisé pour accéder à un membre d'une instance de VARIABLE_LISTS.

Structure lexicale

variable-list-reference, member-identifiant

Les attributs sont spécifiés dans le Tableau 215.

Tableau 215 – Référencement des membres de VARIABLE_LISTS

Usage	Attribut	Description
m	variable-list-reference	Référence à une instance de VARIABLE_LISTS.
m	member-identifiant	Identifiant d'un élément de MEMBERS.

7.38.8 Référencement d'éléments BLOCK_A PARAMETERS**Objet**

Ce référencement est utilisé pour accéder à un élément de l'attribut PARAMETERS d'une instance de BLOCK_A.

Structure lexicale

PARAM, member-identifiant

Les attributs sont spécifiés dans le Tableau 216.

Tableau 216 – Référencement des membres de BLOCK_A PARAMETERS

Usage	Attribut	Description
m	member-identifiant	Identifiant d'un élément de MEMBERS.

7.38.9 Référencement d'éléments de BLOCK_A PARAMETER_LISTS**Objet**

Ce référencement est utilisé pour accéder à un élément de l'attribut PARAMETER_LISTS d'une instance de BLOCK_A.

Structure lexicale

PARAM_LIST, member-identifiant

Les attributs sont spécifiés dans le Tableau 217.

Tableau 217 – Référencement des membres de BLOCK_A PARAMETER_LISTS

Usage	Attribut	Description
m	member-identifiant	Identifiant d'un élément de MEMBERS.

7.38.10 Référencement d'éléments de BLOCK_A LOCAL_PARAMETERS**Objet**

Ce référencement est utilisé pour accéder à un élément de l'attribut LOCAL_PARAMETERS d'une instance de BLOCK_A.

Structure lexicale

LOCAL_PARAM, member-identifiant

Les attributs sont spécifiés dans le Tableau 218.

Tableau 218 – Référencement des membres de BLOCK_A LOCAL_PARAMETERS

Usage	Attribut	Description
m	member-identifiant	Identifiant d'un élément de MEMBERS.

7.38.11 Référencement de BLOCK_A CHARACTERISTICS**Objet**

Ce référencement est utilisé pour accéder aux CHARACTERISTICS (caractéristiques) d'une instance de BLOCK_A.

Structure lexicale

BLOCK, characteristics-identifiant

L'attribut est spécifié dans le Tableau 219.

Tableau 219 – Référencement de BLOCK_A CHARACTERISTICS

Usage	Attribut	Description
m	characteristics-identifiant	Identifiant de la référence de CHARACTERISTICS.

7.38.12 Référencement des membres d'un FILE**Objet**

Ce référencement est utilisé pour accéder à un membre d'une instance de FILE (fichier).

Structure lexicale

file-reference, member-identifiant

Les attributs sont spécifiés dans le Tableau 220.

Tableau 220 – Référencement des membres de FILE

Usage	Attribut	Description
m	file-reference	Référence à une instance de FILE.
m	member-identifiant	Identifiant d'un élément de MEMBERS.

7.38.13 Référencement d'éléments de LIST

Objet

Ce référencement est utilisé pour accéder à un élément d'une instance de LIST.

Structure lexicale

list-reference, (expression, FIRST, LAST)

Les attributs sont spécifiés dans le Tableau 221.

Tableau 221 – Référencement d'éléments de LIST

Usage	Attribut	Description
m	list-reference	Référence à une instance de LIST.
s	expression	Expression qui doit être égale à un entier non négatif qui est dans l'intervalle d'index de l'attribut LIST (pour la description des expressions, voir 7.39.7).
s	FIRST	Premier élément de la liste.
s	LAST	Dernier élément de la liste.

7.38.14 Référencement des membres de CHART

Objet

Ce référencement est utilisé pour accéder à un membre d'une instance de CHART (diagramme).

Structure lexicale

chart-reference, member-identifiant

Les attributs sont spécifiés dans le Tableau 222.

Tableau 222 – Référencement des membres de CHART

Usage	Attribut	Description
m	chart-reference	Référence à une instance de CHART.
m	member-identifiant	Identifiant d'un élément de MEMBERS.

7.38.15 Référencement de membres d'un GRAPH

Objet

Ce référencement est utilisé pour accéder à un membre d'une instance de GRAPH.

Structure lexicale

graph-reference, member-identifiant

Les attributs sont spécifiés dans le Tableau 223.

Tableau 223 – Référencement des membres de GRAPH

Usage	Attribut	Description
m	graph-reference	Référence à une instance de GRAPH.
m	member-identifiant	Identifiant d'un élément de MEMBERS.

7.38.16 Référencement des membres d'une SOURCE

Objet

Ce référencement est utilisé pour accéder à un membre d'une instance de GRAPH.

Structure lexicale

source-reference, member-identifiant

Les attributs sont spécifiés dans le Tableau 224.

Tableau 224 – Référencement des membres de SOURCE

Usage	Attribut	Description
m	source-reference	Référence à une instance de SOURCE.
m	member-identifiant	Identifiant d'un élément de MEMBERS.

7.38.17 Référencement d'AXIS de GRAPH, SOURCE, WAVEFORM

Objet

Ce référencement est utilisé pour accéder à un AXIS (axe) de GRAPH (graphique), SOURCE ou WAVEFORM (courbe).

Structure lexicale

reference, (X_AXIS, Y_AXIS)

Les attributs sont spécifiés dans le Tableau 225.

Tableau 225 – Référencement d'AXIS de GRAPH, SOURCE, WAVEFORM

Usage	Attribut	Description
m	reference	Référence à une instance de GRAPH, une référence à une instance de SOURCE, ou une référence à une instance de WAVEFORM.
c/o	X_AXIS	Utilisé uniquement avec GRAPH; Y_AXIS utilisé dans tous les autres cas.
c/o	Y_AXIS	Utilisé uniquement avec SOURCE et WAVEFORM; X_AXIS utilisé dans tous les autres cas.

7.38.18 Référencement de PARAMETERS d'une instance spécifique de BLOCK_A

Objet

Ce référencement est utilisé pour accéder à un membre de l'attribut PARAMETERS d'une instance spécifique de BLOCK_A.

Structure lexicale

reference, expression, PARAM, member-identifiant

Les attributs sont spécifiés dans le Tableau 226.

Tableau 226 – Référencement de PARAMETERS d'une instance spécifique de BLOCK_A

Usage	Attribut	Description
m	reference	Référence à une instance de BLOCK_A.
m	expression	Entier non négatif qui spécifie l'occurrence de l'instance de BLOCK_A dans le dispositif.
m	member-identifiant	Identifiant d'un élément de MEMBERS.

7.38.19 Référencement de LOCAL_PARAMETERS d'une instance spécifique de BLOCK_A

Objet

Ce référencement est utilisé pour accéder à un membre de l'attribut LOCAL_PARAMETERS d'une instance spécifique de BLOCK_A.

Structure lexicale

reference, expression, LOCAL_PARAM, member-identifiant

Les attributs sont spécifiés dans le Tableau 227.

Tableau 227 – Référencement de LOCAL_PARAMETERS d'une instance spécifique de BLOCK_A

Usage	Attribut	Description
m	reference	Référence à une instance de BLOCK_A.
m	expression	Entier non négatif qui spécifie l'occurrence de l'instance de BLOCK_A dans le dispositif.
m	member-identifiant	Identifiant d'un élément de MEMBERS.

7.38.20 Référencement de CHARACTERISTICS d'une instance spécifique de BLOCK_A

Objet

Ce référencement est utilisé pour accéder à un membre de l'attribut CHARACTERISTICS d'une instance spécifique de BLOCK_A.

Structure lexicale

reference, expression, BLOCK, member-identifiant

Les attributs sont spécifiés dans le Tableau 228.

Tableau 228 – Référencement de CHARACTERISTICS d'une instance spécifique de BLOCK_A

Usage	Attribut	Description
m	reference	Référence à une instance de BLOCK_A.
m	expression	Entier non négatif qui spécifie l'occurrence de l'instance de BLOCK_A dans le dispositif.
m	member-identifiant	Identifiant d'un élément de MEMBERS.

7.38.21 Référencement de CHARTS d'une instance spécifique de BLOCK_A

Objet

Ce référencement est utilisé pour accéder à un membre de l'attribut CHARTS d'une instance spécifique de BLOCK_A.

Structure lexicale

reference, expression, CHART, member-identifiant

Les attributs sont spécifiés dans le Tableau 229.

Tableau 229 – Référencement de CHARTS d'une instance spécifique de BLOCK_A

Usage	Attribut	Description
m	reference	Référence à une instance de BLOCK_A.
m	expression	Entier non négatif qui spécifie l'occurrence de l'instance de BLOCK_A dans le dispositif.
m	member-identifiant	Identifiant d'un élément de MEMBERS.

7.38.22 Référencement de LISTS d'une instance spécifique de BLOCK_A

Objet

Ce référencement est utilisé pour accéder à un membre de l'attribut LISTS d'une instance spécifique de BLOCK_A.

Structure lexicale

reference, expression, LIST, member-identifiant

Les attributs sont spécifiés dans le Tableau 230.

Tableau 230 – Référencement de LISTS d'une instance spécifique de BLOCK_A

Usage	Attribut	Description
m	reference	Référence à une instance de BLOCK_A.
m	expression	Entier non négatif qui spécifie l'occurrence de l'instance de BLOCK_A dans le dispositif.
m	member-identifiant	Identifiant d'un élément de MEMBERS.

7.38.23 Référencement de GRAPHS d'une instance spécifique de BLOCK_A

Objet

Ce référencement est utilisé pour accéder à un membre de l'attribut GRAPHS d'une instance spécifique de BLOCK_A.

Structure lexicale

reference, expression, GRAPH, member-identifiant

Les attributs sont spécifiés dans le Tableau 231.

Tableau 231 – Référencement de GRAPHS d'une instance spécifique de BLOCK_A

Usage	Attribut	Description
m	reference	Référence à une instance de BLOCK_A.
m	expression	Entier non négatif qui spécifie l'occurrence de l'instance de BLOCK_A dans le dispositif.
m	member-identifiant	Identifiant d'un élément de MEMBERS.

7.38.24 Référencement de GRIDS d'une instance spécifique de BLOCK_A

Objet

Ce référencement est utilisé pour accéder à un membre de l'attribut GRIDS d'une instance spécifique de BLOCK_A.

Structure lexicale

reference, expression, GRID, member-identifiant

Les attributs sont spécifiés dans le Tableau 232.

Tableau 232 – Référencement de GRIDS d'une instance spécifique de BLOCK_A

Usage	Attribut	Description
m	reference	Référence à une instance de BLOCK_A.
m	expression	Entier non négatif qui spécifie l'occurrence de l'instance de BLOCK_A dans le dispositif.
m	member-identifiant	Identifiant d'un élément de MEMBERS.

7.38.25 Référencement de MENUS d'une instance spécifique de BLOCK_A

Objet

Ce référencement est utilisé pour accéder à un membre de l'attribut MENUS d'une instance spécifique de BLOCK_A.

Structure lexicale

reference, expression, MENU, member-identifiant

Les attributs sont spécifiés dans le Tableau 233.

Tableau 233 – Référencement de MENUS d'une instance spécifique de BLOCK_A

Usage	Attribut	Description
m	reference	Référence à une instance de BLOCK_A.
m	expression	Entier non négatif qui spécifie l'occurrence de l'instance de BLOCK_A dans le dispositif.
m	member-identifiant	Identifiant d'un élément de MEMBERS.

7.38.26 Référencement de METHODS d'une instance spécifique de BLOCK_A

Objet

Ce référencement est utilisé pour accéder à un membre de l'attribut METHODS d'une instance spécifique de BLOCK_A.

Structure lexicale

reference, expression, METHOD, member-identifiant

Les attributs sont spécifiés dans le Tableau 234.

Tableau 234 – Référencement de METHODS d'une instance spécifique de BLOCK_A

Usage	Attribut	Description
m	reference	Référence à une instance de BLOCK_A.
m	expression	Entier non négatif qui spécifie l'occurrence de l'instance de BLOCK_A dans le dispositif.
m	member-identifiant	Identifiant d'un élément de MEMBERS.

7.38.27 Référencement des'instances de COMPONENT**Objet**

Ce référencement est utilisé pour référencer une instance qui est liée à une COMPONENT_RELATION d'un composant. Si la référence est utilisée dans une déclaration de MENU et si l'instance n'existe pas, l'élément référencé sera traité comme étant non valide. Si la référence est utilisée dans une DECLARATION de METHOD et si l'instance n'existe pas, la METHOD est interrompue. Une telle référence n'est pas autorisée à être utilisée dans une expression conditionnelle.

Structure lexicale

reference, expression, identifier

Les attributs de référencement de composant sont spécifiés dans le Tableau 235.

Tableau 235 – Référencement d'une instance de COMPONENT

Usage	Attribut	Description
o	reference	Référence à un COMPONENT.
o	expression	Expression qui doit être égale à un entier non négatif qui est dans l'intervalle d'index du COMPONENT_RELATION.
o	identifier	Identifiant d'une construction de base du COMPONENT référencé.

7.38.28 Référencement de types de COMPONENT**Objet**

Ce référencement est utilisé pour référencer un type de composant. Toutes les expressions conditionnelles sont évaluées avec des INITIAL_VALUE (valeur initiale) ou, si celles-ci ne sont pas définies, des DEFAULT_VALUE (valeur par défaut) ou avec des valeurs par défaut de type de données.

Structure lexicale

reference, identifier

Les attributs de référencement de composant sont spécifiés dans le Tableau 236.

Tableau 236 – Référencement d'un type COMPONENT

Usage	Attribut	Description
o	reference	Référence à un COMPONENT.
o	identifier	Identifiant d'une construction de base du COMPONENT référencé.

7.39 Chaînes

7.39.1 Spécification d'une chaîne sous forme de littéral chaîne

Objet

Une chaîne de texte peut être spécifiée sous la forme d'un littéral chaîne (voir Annexe A.2.3).

Structure lexicale

`string`

L'attribut est spécifié dans le Tableau 237.

Tableau 237 – Chaîne sous forme de littéral chaîne

Usage	Attribut	Description
m	string	Littéral chaîne

7.39.2 Spécification d'une chaîne en tant que variable chaîne

Objet

Une chaîne de texte spécifiée en tant que variable chaîne est la valeur de la variable chaîne.

Structure lexicale

`reference`

L'attribut est spécifié dans le Tableau 238.

Tableau 238 – Chaîne en tant que variable chaîne

Usage	Attribut	Description
m	reference	Référence à une instance de VARIABLE.

7.39.3 Spécification d'une chaîne en tant que valeur d'énumération

Objet

Une chaîne de valeur d'énumération est une chaîne associée à une des valeurs d'une variable d'énumération.

Structure lexicale

`reference, value`

Les attributs sont spécifiés dans le Tableau 239.

Tableau 239 – Chaîne en tant que valeur d'énumération

Usage	Attribut	Description
m	reference	Référence à une instance de VARIABLE de TYPE ENUMERATION.
m	value	Valeur correspondante au texte de description dans la liste d'énumération.

EXEMPLE La chaîne de valeur d'énumération suivante spécifie la chaîne associée à la valeur 4 de la variable `units_code`: `units_code (4)`

7.39.4 Spécification d'une chaîne en tant que référence de dictionnaire

Objet

Une référence de dictionnaire pointe vers une chaîne dans un dictionnaire de texte.

Structure lexicale

reference

L'attribut est spécifié dans le Tableau 240.

Tableau 240 – Chaîne en tant que référence de dictionnaire

Usage	Attribut	Description
m	reference	Référence à une ou plusieurs chaînes spécifiées dans un dictionnaire (voir 7.41).

7.39.5 Référencement d'attributs HELP et LABEL d'instances d'EDD

Objet

Pour obtenir la chaîne actuelle de LABEL ou HELP d'une instance d'EDD, l'instance et l'attribut LABEL ou HELP doit être référencée.

Structure lexicale

reference, [HELP, LABEL]

Les attributs sont spécifiés dans le Tableau 241.

Tableau 241 – Référencement des attributs HELP et LABEL des instances de l'EDD

Usage	Attribut	Description
m	reference	Instance d'EDD qui contient les attributs LABEL et/ou HELP.
s	HELP	Chaîne HELP de l'instance.
s	LABEL	Chaîne LABEL de l'instance.

7.39.6 Opérations de chaîne

Objet

Les chaînes littérales, les variables chaînes, les valeurs d'énumération chaîne, les références de dictionnaire et les valeurs d'attribut de constructions de base peuvent être concaténées les unes avec les autres. Si un terme contient des références de types de données numériques, les valeurs doivent être converties en chaîne par l'application EDD.

EXEMPLE variable_de_type_chaine = "Description"

"Electronic" + "Device" + variable_de_type_chaine conduit à "Electronic Device Description".

Structure lexicale

(string)<exp>, [(concatenation-operator, string)<exp>]+

Les attributs sont spécifiés dans le Tableau 242.

Tableau 242 – Opération de chaîne

Usage	Attribut	Description
m	string	Chaînes littérales, variables chaînes, valeurs d'énumération de chaîne, références de dictionnaire ou référence de HELP, LABEL.
o	concatenation-operator	Les chaînes doivent être concaténées.

7.39.7 Formats de chaîne d'invite de commande

Objet

Des chaînes d'invite de commande peuvent être utilisées pour incorporer des éléments formatés dans des chaînes. Ces éléments sont composés d'un spécificateur de format facultatif et d'un VARIABLE ID ou une référence.

Structure lexicale

```
format, (reference, id)
label, (reference, id)
```

Les attributs sont spécifiés dans le Tableau 243.

Tableau 243 – Spécificateur de format

Usage	Attribut	Description
m	format	Correspond à la chaîne de format printf ANSI C (voir l'ISO/CEI 9899). Si le format n'est pas spécifié, le format de l'instance de VARIABLE référencée est utilisé. Si aucun format n'est spécifié, un format par défaut est utilisé sur la base du type d'instance de VARIABLE.
m	label	LABEL de l'instance de VARIABLE.
o/s	reference	Valeur ou étiquette de l'instance de VARIABLE référencée.
o/s	id	ID d'élément des variables.

7.40 Expression

7.40.1 Structure générale

Objet

Il existe trois types d'expressions.

- Les expressions primaires comprennent des constantes, des expressions entre parenthèses, des références de VARIABLE, des accès d'élément d'index ou un accès au statut d'instances de VARIABLE. Ces opérations sont associatives par la gauche, c'est-à-dire qu'elles sont évaluées de gauche à droite.
- Des expressions unaires comprennent tous les opérateurs évaluant un opérande unique. Ces opérations sont associatives par la droite, c'est-à-dire qu'elles sont évaluées de droite à gauche.
- Les expressions binaires comprennent tous les opérateurs évaluant deux opérandes. Ces opérations sont associatives par la gauche, c'est-à-dire qu'elles sont évaluées de gauche à droite.

7.40.2 Expressions primaires

Le Tableau 244 ci-dessous résume les expressions primaires dans le langage EDDL.

Tableau 244 – Expressions primaires

Expression primaire	Description
constante	Expression dont la valeur est identique à la valeur de la constante.
expression entre parenthèses	Expression dont la valeur est identique à la valeur de l'expression incluse.
référence de variable	Expression dont la valeur est la valeur de la VARIABLE référencée.
ARRAY_INDEX	Expression dont la valeur est dérivée du numéro d'index du VALUE_ARRAY référençant.
valeurs d'attribut de VARIABLE	Référence constituée de l'identifiant d'une instance de VARIABLE et d'un attribut. La référence fournit la valeur correspondante de l'attribut (voir Tableau 245 pour les valeurs retournées).
CURRENT_ROLE	Constante chaîne qui contient le rôle actuel. La valeur de CURRENT_ROLE (rôle actuel) doit être prise en charge par l'application EDD. Les valeurs pour la constante chaîne doivent être définies par l'attribut ROLE de MENU.
PI	Constante mathématique π .
valeurs d'attribut d'un AXIS	Référence à une instance et un attribut d'AXIS. La référence fournit la valeur correspondante des attributs (voir Tableau 246 pour les valeurs retournées). Ces attributs sont en lecture seule sauf indication contraire.
valeurs d'attribut d'une LIST	Référence à une instance et un attribut de LIST. La référence fournit la valeur correspondante des attributs (voir Tableau 247 pour les valeurs retournées). Ces attributs sont en lecture seule sauf indication contraire.

Tableau 245 – Valeurs d'attribut de VARIABLE

Attributs	Type de données	Description
MIN_VALUE	type d'instance de VARIABLE	Valeur qui est conforme à la MIN_VALUE de l'instance de VARIABLE. Si plus d'une MIN_VALUE est définie, MIN_VALUE plus l'identifiant d'intervalle (voir Tableau 165) doit être utilisé pour le référencement.
MAX_VALUE	type d'instance de VARIABLE	Valeur qui est conforme à la MAX_VALUE de l'instance de VARIABLE. Si plus d'une MAX_VALUE est définie, MAX_VALUE plus l'identifiant d'intervalle (voir Tableau 165) doit être utilisé pour le référencement.
SCALING_FACTOR	double	Valeur qui est conforme au SCALING_FACTOR de l'instance de VARIABLE.
DEFAULT_VALUE	type d'instance de VARIABLE	Valeur qui est conforme à la DEFAULT_VALUE de l'instance de VARIABLE.
INITIAL_VALUE	type d'instance de VARIABLE	Valeur qui est conforme à l'INITIAL_VALUE de l'instance de VARIABLE.
VARIABLE_STATUS		Valeur qui indique le statut de la VARIABLE. Les statuts suivants sont définis: VARIABLE_STATUS_NONE spécifie que la valeur est valide et non un des autres statuts. VARIABLE_STATUS_INVALID spécifie que la valeur est hors de l'intervalle. VARIABLE_STATUS_NOT_ACCEPTED spécifie que la valeur n'a pas pu être transférée. VARIABLE_STATUS_NOT_SUPPORTED spécifie que la valeur n'est pas prise en charge. VARIABLE_STATUS_CHANGED spécifie que la valeur a été modifiée par l'utilisateur ou par une méthode. VARIABLE_STATUS_LOADED spécifie que la valeur a été chargée avec succès. VARIABLE_STATUS_INITIAL spécifie que la valeur est inchangée. VARIABLE_STATUS_NOT_CONVERTED spécifie que la valeur n'a pas été convertie, par exemple, parce que les unités ne sont pas compatibles.

Tableau 246 – Valeurs d'attribut d'AXIS

Attributs	Type de données	Description
MIN_VALUE	type arithmétique	Valeur qui est conforme à la MIN_VALUE de l'instance d'AXIS.
MAX_VALUE	type arithmétique	Valeur qui est conforme à la MAX_VALUE de l'instance d'AXIS.
SCALING	LINEAR (linéaire) ou LOGARITHMIC (logarithmique)	Valeur qui est conforme au SCALING (mise à l'échelle) de l'instance d'AXIS.
VIEW_MIN	type arithmétique	Valeur qui est conforme au VIEW_MIN de l'instance d'AXIS; VIEW_MIN peut être lu et écrit.
VIEW_MAX	type arithmétique	Valeur qui est conforme au VIEW_MAX de l'instance d'AXIS; VIEW_MAX peut être lu et écrit.

Tableau 247 – Valeurs d'attribut de LIST

Attributs	Type de données	Description
CAPACITY	unsigned_integer	Valeur qui est conforme à la CAPACITY (capacité) de l'instance de LIST.
COUNT	unsigned_integer	Valeur qui est conforme au COUNT (compteur) de l'instance de LIST.

7.40.3 Expressions unaires

Une expression unaire est constituée d'un opérande numérique, une expression, précédée par un opérateur unaire. Le Tableau 248 spécifie les expressions unaires dans le langage EDDL.

Tableau 248 – Expressions unaires

Opérateur	Description
-	Négation arithmétique de l'opérande.
~	Négation binaire de l'opérande, c'est-à-dire que chaque bit du résultat est l'inverse du bit correspondant de l'opérande. L'opérande de l'opérateur ~ doit avoir une valeur entière.
!	Négation logique de l'opérande.

7.40.4 Expressions binaires

7.40.4.1 Structure générale

Une expression binaire est constituée de deux opérandes ou expressions, séparés par un opérateur binaire. Si un opérande a une valeur à virgule flottante, l'autre opérande est converti (promu) en valeur à virgule flottante. Cette sous-section spécifie les types d'expressions binaires suivants.

- Multiplicatif
- Additif
- Décalage
- Relationnel
- Égalité
- ET binaire (&)
- XOU binaire (^)
- OU binaire (|)

- ET logique (&&)
- OU logique (||)
- Évaluation conditionnelle

7.40.4.2 Opérateurs multiplicatifs

Les opérateurs multiplicatifs spécifient la multiplication et la division d'opérandes. Le Tableau 249 spécifie les opérateurs multiplicatifs.

Tableau 249 – Opérateurs multiplicatifs

Opérateur	Description
*	Multiplication de ses opérandes.
/	Division du premier opérande par le second opérande. Le résultat de l'opérateur / est le quotient de la division.
%	Division du premier opérande par le second opérande. Le résultat de l'opérateur % est le résidu.

7.40.4.3 Opérateurs additifs

Les opérateurs additifs spécifient l'addition et la soustraction d'opérandes numériques. Si cet opérateur est appliqué à des opérandes numériques, le résultat est alors la somme de ses deux opérandes, et l'opération est commutative. Si l'opérateur + est appliqué à des opérandes chaînes, le résultat est une valeur de chaîne générée par ajout d'une seconde chaîne à la première chaîne et l'opération n'est pas commutative.

L'opérateur moins indique une soustraction. La valeur résultante de cette opération est la différence entre ses opérandes. Le second opérande est soustrait du premier. Le Tableau 250 spécifie des opérateurs additifs.

Tableau 250 – Opérateurs additifs

Opérateur	Description
+	Addition de ses opérandes.
-	Soustraction du second opérande du premier.

7.40.4.4 Opérateurs de décalage

Les opérateurs << et >> spécifient un décalage du premier opérande du nombre de bits spécifié par le second opérande. Les opérandes des opérateurs << et >> doivent avoir des valeurs entières. Le Tableau 251 spécifie les opérateurs de décalage.

Tableau 251 – Opérateurs de décalage

Opérateur	Description
<<	Premier opérande sur la gauche. Les bits décalés à l'extérieur sont rejetés, et les bits laissés vacants sont remplis avec des zéros.
>>	Premier opérande sur la droite. Les bits décalés à l'extérieur sont rejetés. Si le premier opérande est inférieur à 0, les bits laissés vacants sont remplis avec des 1, sinon ils sont remplis avec des zéros.

7.40.4.5 Opérateurs relationnels

Les opérateurs relationnels (<, <=, >, >=) spécifient une comparaison des opérandes. Le résultat de ce type d'expression est 1 si la relation testée est vraie; sinon, le résultat est 0. Les opérandes doivent être des types de données numériques. Le Tableau 252 spécifie les opérateurs relationnels.

Tableau 252 – Opérateurs relationnels

Opérateur	Description
<	Teste la relation "inférieur à".
<=	Teste la relation "inférieur ou égal à".
>	Teste la relation "supérieur à".
>=	Teste la relation "supérieur ou égal à".

7.40.4.6 Opérateurs d'égalité

Les opérateurs d'égalité sont == et !=. Le résultat de ce type d'expression est 1 si la relation testée est vraie; sinon, le résultat est 0. Les opérateurs doivent être du même type de données. Tous les types de données sont autorisés. Le Tableau 253 spécifie les opérateurs d'égalité.

Tableau 253 – Opérateurs d'égalité

Opérateur	Description
==	Teste la relation "est égal à".
!=	Teste la relation "est différent de".

7.40.4.7 Opérateur ET binaire (&)

L'opérateur & spécifie le ET binaire de ses opérandes entiers, c'est-à-dire que chaque bit du résultat est défini si chacun des bits correspondants des opérandes est défini. Les opérandes de l'opérateur & doivent avoir des valeurs entières.

7.40.4.8 Opérateur XOU binaire (^)

L'opérateur ^ spécifie le OU exclusif binaire de ses opérandes entiers, c'est-à-dire que chaque bit du résultat est défini si un seul des bits correspondants des opérandes est défini. Les opérandes de l'opérateur ^ doivent avoir des valeurs entières.

7.40.4.9 Opérateur OU binaire (|)

L'opérateur | spécifie le OU inclusif binaire de ses opérandes entiers, c'est-à-dire que chaque bit du résultat est défini si l'un des bits correspondants des opérandes est défini. Les opérandes de l'opérateur | doivent avoir des valeurs entières.

7.40.4.10 Opérateur ET logique (&&)

L'opérateur && spécifie l'évaluation ET booléenne de ses opérandes entiers. Si un opérande fait référence à une VARIABLE non valide (VALIDITY égal à FALSE), l'opérande doit être évalué à 0. Le résultat de ce type d'expression est 1 si les deux opérandes ne sont pas égaux à 0; sinon, le résultat est 0. Si le premier opérande est égal à 0, le second opérande n'est pas évalué.

7.40.4.11 Opérateur OU logique (||)

L'opérateur || spécifie l'évaluation OU booléenne de ses opérandes entiers. Si un opérande fait référence à une VARIABLE non valide (VALIDITY égal à FALSE), l'opérande doit être évalué à 0. Le résultat de ce type d'expression est 1 si l'un des opérandes n'est pas égal à 0; sinon, le résultat est 0. Si le premier opérande n'est pas égal à 0, le second opérande n'est pas évalué.

7.40.4.12 Évaluation conditionnelle

Les opérateurs ? et : sont utilisés pour une évaluation conditionnelle.

EXEMPLE 1

```
result = logexpr? trueexpr: falseexpr;
logexpr est évalué et s'il est différent de zéro, le résultat est la valeur de trueexpr. Sinon, falseexpr est exécuté.
```

EXEMPLE 2 La structure if-else suivante a une logique identique à l'exemple 1:

```
if (logexpr)
  result = trueexpr;
else
  result = falseexpr;
```

7.41 Dictionnaire de texte

Structure générale

Le dictionnaire est une collection de chaînes de textes qui peuvent être utilisées dans une EDD. Le texte du dictionnaire est référencé dans une EDD. Le dictionnaire est accessible depuis n'importe quelle EDD afin d'assurer la cohérence des chaînes de texte (par exemple, pour utilisation dans des attributs de LABEL et HELP).

Généralement, un dictionnaire de texte commun est utilisé pour un groupe de dispositifs types. De plus, un dictionnaire spécifique pour l'utilisateur peut être spécifié. Le dictionnaire de texte prend en charge des textes multilingues.

Structure lexicale

```
identifier (language-code, country-code, string)+
```

Les attributs sont spécifiés dans le Tableau 254.

Tableau 254 – Attributs de dictionnaire de texte

Usage	Attribut	Description
m	identifier	Nom d'une chaîne de texte dans une EDD.
m	language-code	Langue utilisée et présentation ASCII ou multicode d'une chaîne; les codes de langue sont définis dans l'ISO 639.
o	country-code	Pays dans lequel la langue est parlée; les codes de pays sont définis dans l'ISO 3166-1.
m	string	Chaîne de texte dans la langue sélectionnée.

En plus des codes de pays normalisés définis dans l'ISO 3166-1, le code pays ZZ peut être utilisé pour spécifier une version courte d'une chaîne. Des versions courtes de chaînes sont utiles pour les applications EDD exécutées sur des dispositifs mobiles qui ont une taille d'écran limitée.

EXEMPLE

write_protected
"Write Protected"
"|de|Schreibgeschützt"
"|fr|Protégé en écriture"
"|it|Protetto in scrittura"
"|es|Protegido contra escritura"
"|en ZZ|Wr Prot"
"|de ZZ|Geschützt"

L'identifiant "write_protected" est l'identifiant qui est référencé dans l'EDD. Les qualificateurs |de|, |fr|, |it|, |es| spécifient le texte pour les langues Allemand, Français, Italien et Espagnol. Les qualificateurs "en ZZ" et "de ZZ" spécifient des versions courtes des chaînes anglaise et allemande. La même approche peut également être utilisée pour fournir des versions courtes pour le français, l'italien et l'espagnol.

Annexe A (normative)

Définition de format EDDL

A.1 Préprocesseur EDDL

A.1.1 Structure générale

Objet

Le préprocesseur génère une EDD avant compilation. Le préprocesseur remplace des parties du texte d'EDD, insère le contenu d'autres fichiers ou supprime le traitement des parties de l'EDD en supprimant des sections.

Structure

Toutes les directives du préprocesseur démarrent avec un symbole de hachure (#) comme premier caractère sur une ligne. Un espace (caractères ESPACE ou TAB) peut apparaître après le # initial pour obtenir une indentation correcte.

A.1.2 Directives

A.1.2.1 #define

Objet

La directive #define donne un nom significatif à une constante dans l'EDD.

Structure

```
#define identifiant token-string
```

Cette forme de directive #define donne l'instruction au préprocesseur de remplacer toutes les occurrences suivantes de l'identifiant dans le fichier de l'EDD par token-string.

NOTE L'identifiant n'est pas remplacé s'il apparaît dans un commentaire, dans une chaîne, ou dans un autre identifiant plus long.

```
#define identifiant
```

Cette forme de directive #define donne l'instruction au préprocesseur de supprimer toutes les occurrences de l'identifiant dans le fichier EDD. L'identifiant reste défini et peut être testé en utilisant les directives #if defined et #ifdef.

```
#define identifiant (argument, ... ,argument) token-string
```

Cette forme de la directive #define donne l'instruction au préprocesseur de créer des macros de type fonction. Cette forme accepte une liste d'arguments qui doivent apparaître entre parenthèses et sont séparés par des virgules.

Lorsqu'une macro a été définie sous cette forme de syntaxe, les instances textuelles consécutives suivies par une liste d'arguments constituent un appel de macro. Les arguments réels suivant une instance d'identifiant dans le fichier source sont mis en correspondance avec les paramètres formels correspondants dans la définition de la macro. Chaque paramètre formel dans token-string est remplacé par l'argument réel correspondant. Toutes les macros dans l'argument réel sont développées avant que la directive remplace le paramètre formel.

Chaque argument dans la liste doit être unique.
Aucun espace n'est autorisé pour séparer un identifiant et la parenthèse d'ouverture.
Pour les directives longues sur des lignes sources multiples, une barre oblique inverse (\) doit être utilisée avant le caractère NEWLINE (saut de ligne).
Aucun espace n'est autorisé entre le nom et "(".

A.1.2.2 **#include**

Objet

La directive **#include** indique au préprocesseur de traiter le contenu d'un fichier spécifié comme si ce contenu était apparu dans le programme source au point où la directive apparaît.

Structure

```
#include "filename"
#include <filename>
```

Lire le contenu du fichier nommé avec le nom de fichier. Le préprocesseur traite ces données comme si elles faisaient partie du fichier actuel.

Les deux formes de syntaxe causent le remplacement de cette directive par le contenu entier du fichier inclus spécifié. La différence entre les deux formes est l'ordre dans lequel le préprocesseur recherche les fichiers inclus lorsque le chemin n'est pas complètement spécifié.

Aucun jeton supplémentaire n'est autorisé sur la ligne de directive après le " ou > final.

A.1.2.3 **#line**

Objet

La directive **#line** conduit le préprocesseur à remplacer le numéro de ligne et le nom de fichier stockés en interne par le compilateur par un numéro de ligne et un nom de fichier donnés. Le processeur de l'EDD utilise le numéro de ligne et le nom de fichier pour faire référence aux erreurs qu'il rencontre pendant le traitement. Le numéro de ligne fait généralement référence à la ligne d'entrée actuelle, et le nom du fichier fait référence au fichier d'entrée actuel. Le numéro de ligne est incrémenté une fois que chaque ligne est traitée.

Structure

```
#line integer-constant "filename"
```

integer-constant et filename conduisent le préprocesseur à remplacer le numéro de ligne stocké en interne par le processeur de l'EDD par un numéro de ligne et un nom de fichier donnés. Integer-constant est interprété comme étant le numéro de ligne de la ligne suivante et est incrémenté une fois que chaque ligne est traitée.

A.1.2.4 **#if, #elif, #else, et #endif**

Objet

La directive **#if**, avec les directives **#elif**, **#else**, et **#endif**, contrôle le traitement de parties d'un fichier EDD.

```
#if constant-expression
```

Les lignes suivantes jusqu'à la directive **#else**, **#elif** ou **#endif**, apparaissent dans la sortie uniquement si constant-expression produit une valeur différente de zéro. Les opérateurs binaires "&&" et "||" sont autorisés dans constant-expression ainsi que l'opérateur "?:" et les opérateurs unaires "-", "!" et "~".

De plus, l'opérateur unaire défini peut être utilisé dans des expressions constantes spéciales, comme décrit par la syntaxe suivante:

```
defined(identifiant)
defined identifiant
```

Cette expression constante est considérée comme étant vraie (différente de zéro) si l'identifiant est actuellement défini; sinon, la condition est fausse. Un identifiant défini comme un texte vide est considéré comme étant défini. La directive définie peut être utilisée dans une directive `#if` et `#elif`, mais pas ailleurs.

```
#elif constant-expression
```

Un nombre quelconque de directives `#elif` peuvent apparaître entre une directive `#if`, `#ifdef` ou `#ifndef` et une directive `#else` ou `#endif` correspondante. Les lignes suivant la directive `#elif` apparaissent dans la sortie uniquement si toutes les conditions suivantes sont vraies.

- La constant-expression dans les directives `#if` précédentes est évaluée à zéro, le nom dans le `#ifdef` précédent n'est pas défini ou le nom dans les directives `#ifndef` précédentes est défini.
- Les constant-expression (expressions constantes) dans toutes les directives `#elif` inversées sont évaluées à zéro.
- La constant-expression actuelle est évaluée à pas-zéro.

Si la constant-expression est évaluée à pas-zéro, ignorer les directives `#elif` et `#else` suivantes jusqu'au `#endif` correspondant. Toute constant-expression autorisée dans une directive `#if` est autorisée dans une directive `#elif`.

```
#else
```

Si toutes les occurrences de constant-expression sont fausses, ou si aucune directive `#elif` n'apparaît, le préprocesseur sélectionne le bloc de texte après la clause `#else`. Si la clause `#else` est omise et que toutes les instances de constant-expression dans le bloc `#if` sont fausses, aucun bloc de texte n'est sélectionné.

```
#endif
```

Termine une section de lignes qui commence avec une des directives conditionnelles `#if`, `#ifdef` ou `#ifndef`. Chaque directive doit correspondre à un `#endif`.

NOTE 1 Le préprocesseur reconnaît des paramètres formels pour des macros dans des corps de directive `#define`, même lorsqu'ils se trouvent à l'extérieur de constantes de caractère et de chaînes entre guillemets. Les noms de macro ne sont pas reconnus dans des constantes de caractère ou des chaînes entre guillemets pendant une analyse normale. Par conséquent, `#define aaa bbb` ne développe pas `aaa` en LABEL "aaa".

NOTE 2 Les macros ne sont pas développées en traitant une directive `#define` ou `#undef`. Par conséquent:

```
#define aaa bbb
```

```
#define ccc aaa
```

```
#undef aaa
```

```
ccc
```

```
produit aaa.
```

NOTE 3 Les macros ne sont pas développées pendant l'analyse qui détermine les paramètres réels pour un autre appel de macro.

Par conséquent:

```
#define turn(aaa,bbb) bbb aaa
```

```
#define ccc ddd
```

```
turn(ccc, #define ccc eee)
```

produit #define ddd eee ddd.

A.1.2.5 #ifdef, #ifndef et #undef

Objet

Les directives #ifdef et #ifndef contrôlent le traitement des parties d'un fichier EDD. Les directives #ifdef et #ifndef effectuent la même tâche que la directive #if lorsqu'elle est utilisée avec un identifiant défini.

```
#ifdef identifiant
```

Les lignes suivantes jusqu'à la directive #else, #elif ou #endif correspondante apparaissent dans la sortie uniquement si l'identifiant a été défini. L'identifiant est spécifié en utilisant une directive #define ou à l'extérieur de l'EDD et en l'absence d'une directive #undef intercalée. Aucun jeton additionnel n'est autorisé sur la ligne de directive après le nom.

```
#ifndef identifiant
```

Les lignes suivantes jusqu'à la directive #else, #elif ou #endif correspondante apparaissent dans la sortie uniquement si le nom n'a pas été défini ou si sa définition a été supprimée avec une directive #undef. Aucun jeton additionnel n'est autorisé sur la ligne de directive après le nom.

```
#undef identifiant
```

La directive #undef cause la suppression de la définition d'un identifiant par le préprocesseur. Aucun jeton additionnel n'est autorisé sur la ligne de directive après le nom.

A.1.3 Macros prédéfinies

A.1.3.1 Structure générale

Le préprocesseur doit reconnaître un ensemble spécifié de macros prédéfinies. Il convient que ces macros soient utilisées ailleurs dans l'EDD et sont à titre d'information.

Ces macros n'acceptent aucun argument et ne peuvent pas être redéfinies.

A.1.3.2 Liste de macros prédéfinies

A.1.3.2.1 __FILE__

Objet

__FILE__ contient le nom du fichier EDD actuel.

Structure

```
__FILE__
```

__FILE__ est développé en chaîne entourée par des guillemets doubles.

A.1.3.2.2 __LINE__

Objet

__LINE__ contient le numéro de ligne dans le fichier EDD actuel.

Structure

```
__LINE__
```

__LINE__ est une constante entière décimale. Elle peut être modifiée avec une directive #line.

A.1.4 Caractères NEWLINE

Un caractère NEWLINE termine une constante de caractère ou une chaîne entre guillemets. La barre oblique inverse (\) suivie d'un NEWLINE dans le corps d'une instruction #define est utilisée pour continuer la définition sur la ligne suivante.

A.1.5 Commentaires

Un commentaire est une séquence de caractères commençant par une combinaison barre oblique/astérisque (/*) et se termine par une combinaison astérisque/barre oblique (*). Un commentaire peut comprendre une combinaison quelconque de caractères de l'ensemble de caractères représentable, comprenant des caractères NEWLINE.

De plus, le commentaire à une ligne est pris en charge s'il est précédé par deux barres obliques (//). Un commentaire commençant par deux barres obliques est terminé par le caractère NEWLINE suivant qui n'est pas précédé par un caractère d'échappement.

A.2 Conventions

A.2.1 Constantes entières

Une constante entière peut être spécifiée en notation binaire, octale, décimale ou hexadécimale. Le Tableau A.1 spécifie les conventions pour chaque type de notation.

Tableau A.1 – Conventions pour les constantes entières

Type de constante entière	Conventions
binaire	Séquence non vide de chiffres binaires 0 et 1 précédée par 0b ou 0B.
octale	Séquence non vide des chiffres 0 à 7 commençant par 0.
décimale	Séquence non vide des chiffres décimaux 0 à 9, ne commençant pas par 0.
hexadécimal	Séquence non vide de chiffres hexadécimaux précédée par 0x ou 0X. Les chiffres hexadécimaux sont les chiffres 0 à 9 et les lettres a à f (ou A à F) avec les valeurs 10 à 15, respectivement.

A.2.2 Constantes à virgule flottante

Structure lexicale

Une constante à virgule flottante comprend quatre parties:

- une partie entière, une séquence de chiffres décimaux;
- un point décimal (.);
- une partie fractionnaire, une séquence de chiffres décimaux;
- une partie exposant, une séquence éventuellement signée de chiffres décimaux précédée par une des lettres e ou E.

Règles

Les règles suivantes s'appliquent à l'utilisation de constantes à virgule flottante:

- la partie entière ou la partie fractionnaire peut être omise, mais pas les deux;
- le point décimal ou la partie exposant peut être omis(e), mais pas les deux.

Exemple: Des exemples de constantes à virgule flottante sont présentés ci-dessous:

59,
,87

48,93

4,8e12

Le calcul de l'expression algébrique doit utiliser le type de données le plus précis.

A.2.3 Chaînes littérales

Les chaînes littérales dans une EDD doivent être codées en utilisant l'ISO Latin-1 (ISO/CEI 8859-1) ou UTF-8 (RFC 3629). Dans une EDD donnée, toutes les chaînes littérales doivent utiliser le même codage. Une chaîne littérale est une séquence de caractères éventuellement vide entourée par des guillemets doubles ("). Les caractères entre guillemets doivent être un caractère quelconque sauf les suivants:

- guillemet (");
- barre oblique (\);
- saut de ligne.

En utilisant les séquences d'échappement dans les chaînes littérales, une chaîne constante peut également contenir des séquences d'échappement qui représentent un caractère. Le Tableau A.2 présente les séquences d'échappement et leur résultat.

Tableau A.2 – Utilisation de séquences d'échappement dans des chaînes littérales

Code d'échappement	Résultat
'	Guillemet simple.
"	Guillemet double.
	Barre verticale.
?	Point d'interrogation.
\	Barre oblique inverse.
\a	Alerte.
\f	Saut de page.
\n	Saut de ligne.
\r	Saut de paragraphe.
\t	Tabulation horizontale.
\v	Tabulation verticale.

A.2.4 Utilisation de codes de langue dans des chaînes de constantes

Une chaîne littérale peut encapsuler toutes les traductions d'une phrase donnée. Le code de langue est spécifié en utilisant l'ISO/CEI 8859-1. Si une chaîne littérale ne contient pas des traductions pour toutes les langues, une langue par défaut doit être utilisée. Le Tableau A.3 présente des exemples de codes de langue.

Tableau A.3 – Utilisation de codes de langue dans des chaînes littérales

Code de langue	Langue
aucun code assigné	Langue par défaut, utilisée si une langue est demandée et n'est pas définie.
[zh]	Chinois
[de]	Allemand
[en]	Anglais
[es]	Espagnol
[fr]	Français

Code de langue	Langue
it	Italien
ja	Japonais
ko	Coréen

NOTE L'anglais est recommandé en tant que langue par défaut.

EXEMPLE La chaîne littérale suivante spécifie l'expression anglaise "Invalid selection" en anglais et en allemand:

"Invalid Selection"

"|de|Unzulässige Auswahl"

A.3 Opérateurs

Le Tableau A.4 contient des opérateurs du langage EDDL.

Tableau A.4 – Opérateurs EDDL

!	!=	%	%=
&	&&	&=	(
)	*	*=	+
++	+=	,	-
--	-=	.	/
/=	:	;	<
<<	<<=	<=	=
==	>	>=	>>
>>=	?	[	]
^	^=	{	
=		}	~
->			

A.4 Mots clés

Le Tableau A.5 contient les mots clés EDDL:

Tableau A.5 – Mots clés EDDL

Mot clé EDDL	Mot clé EDDL	Mot clé EDDL	Mot clé EDDL
ACCESS	DOMAINS	MAX_VALUE6	SCAN_LIST
ACTUATOR	double	MAX_VALUE7	SCOPE
ACTUATOR_ELECTRIC	DOUBLE	MAX_VALUE8	SECONDS
ACTUATOR_ELECTRO_PNEUMATIC	DURATION	MAX_VALUE9	SELECT
ACTUATOR_HYDRAULIC	DYNAMIC	MIN_VALUE	SELF
ADD	EDD	MIN_VALUE1	SELF_CORRECTING
ADDRESSING	EDD_PROFILE	MIN_VALUE2	SEMICONDUCTOR
ALARM	EDD_REVISION	MIN_VALUE3	SENSOR
ANALOG_INPUT	EDD_VERSION	MIN_VALUE4	SENSOR_ACOUSTIC
ANALOG_OUTPUT	EDIT_DISPLAY	MIN_VALUE5	SENSOR_ANALYTIC
APPINSTANCE	EDIT_DISPLAY_ITEMS	MIN_VALUE6	SENSOR_ANALYTIC_GAS

Mot clé EDDL	Mot clé EDDL	Mot clé EDDL	Mot clé EDDL
ARGUMENTS	EDIT_DISPLAYS	MIN_VALUE7	SENSOR_ANALYTIC_LIQUID
ARRAY	EDIT_FORMAT	MIN_VALUE8	SENSOR_CONCENTRATION
ARRAY_INDEX	EDIT_ITEMS	MIN_VALUE9	SENSOR_DENSITY
ARRAYS	ELECTRICAL_DISTRIBUTION	MINIMUM_NUMBER	SENSOR_FLOW
ASCII	ELECTRICAL_DISTRIBUTION_POWER_MONITORING	MINIMUM_NUMBER	SENSOR_FLOW_CORIOLIS
AUTO_CREATE	ELEMENTS	MINUTES	SENSOR_FLOW_ELECTROMAGNETIC
AXES	else	MINUTES	SENSOR_FLOW_MECHANICAL
AXIS	ELSE	MISC	SENSOR_FLOW_RADIOMETRIC
AXIS_ITEMS	EMPHASIS	MISC	SENSOR_FLOW_THERMAL
BAD	ENUMERATED	MISC_ERROR	SENSOR_FLOW_ULTRASONIC
BIT_ENUMERATED	EUC	MISC_ERROR	SENSOR_FLOW_VORTEX_COUNTER
BITSTRING	EVENT	MISC_WARNING	SENSOR_LEVEL
BLOCK	EVERYTHING	MISC_WARNING	SENSOR_LEVEL_BUOYANCY
BLOCKS	EXIT_ACTIONS	MODE	SENSOR_LEVEL_CAPACITIVE
break	FALSE	MODE	SENSOR_LEVEL_ECHO
CAN_DELETE	FF	MODE_ERROR	SENSOR_LEVEL_HYDROSTATIC
CAPACITY	FILE	MODE_ERROR	SENSOR_LEVEL_RADIOMETRIC
case	FILE_ITEMS	MODULE	SENSOR_MULTIVARIABLE
CASE	FILES	MODULE	SENSOR_POSITION
CATALOG	FILTER	MORE	SENSOR_PRESSURE
char	FIRST	MORE	SENSOR_TEMPERATURE
CHARACTERISTICS	float	NETWORK_COMPONENT	SEPARATOR
CHART	FLOAT	NETWORK_COMPONENT	SERVICE
CHART_ITEMS	for	NETWORK_COMPONENT_WIRELESSADAPTER	short
CHARTS	FREQUENCY_CONVERTER	NEXT	SIBLING_COMPONENT
CHECK_CONFIGURATION	FREQUENCY_INPUT	NEXT	signed
CHILD	FREQUENCY_OUTPUT	NEXT_COMPONENT	SIMULATION
CHILD_COMPONENT	FUNCTION	NEXT_COMPONENT	SLOT
CLASS	GAUGE	NO_LABEL	SMALL
CLASSIFICATION	GRAPH	NO_UNIT	SOFTWARE
COLLECTION	GRAPH_ITEMS	NUMBER	SOURCE
COLLECTION_ITEMS	GRAPHS	NUMBER_OF_ELEMENTS	SOURCE_ITEMS
COLLECTIONS	GRID	NUMBER_OF_POINTS	SOURCES
COLUMNBREAK	GRID_ITEMS	OBJECT_REFERENCE	STATE
COMM	GRIDS	OCTET	STRIP
COMMAND	GROUP	OF	STYLE
COMMANDS	HANDLING	OFFLINE	SUB_SLOT
COMPONENT	HARDWARE	ONLINE	SUCCESS
COMPONENT_FOLDER	HART	OPERATE	SUMMARY
COMPONENT_FOLDERS	HEADER	OPERATION	SUPPLIED_INTERFACE
COMPONENT_PARENT	HEIGHT	OPTIONAL	SWEEP
COMPONENT_PATH	HELP	ORIENTATION	switch
COMPONENT_REDEFINITIONS	HIDDEN	OUTPUT	SWITCHGEAR
COMPONENT_REFERENC	HIGH_HIGH_LIMIT	PACKED_ASCII	TABLE

Mot clé EDDL	Mot clé EDDL	Mot clé EDDL	Mot clé EDDL
COMPONENT_REFERENCES	HIGH_LIMIT	PAGE	TEMPLATE
COMPONENT_RELATION	HORIZONTAL	PARAM	TEMPLATES
COMPONENT_RELATIONS	HORIZONTAL_BAR	PARAM_LIST	TEMPORARY
COMPONENTS	HOURS	PARAMETER_LISTS	TEMPS
COMPUTATION	if	PARAMETERS	TIME_FORMAT
CONNECTION	IF	PARENT	TIME_SCALE
CONNECTIONS	IGNORE_IN_HOST	PARENT_COMPONENT	time_t
CONSTANT_UNIT	IGNORE_IN_TEMPORARY_MASTER	PASSWORD	TIME_VALUE
CONTAINED	IMAGE	PATH	TRANSACTION
continue	IMAGE_ITEMS	PHYSICAL	TRANSDUCER
CONTROLLER	IMAGES	PI	TRANSPARENT
CONVERTER	IMPORT	POST_EDIT_ACTIONS	TRUE
CORRECTABLE	INDEX	POST_READ_ACTIONS	TUNE
CORRECTION	INDICATOR	POST_WRITE_ACTIONS	TYPE
COUNT	INFO	PRE_EDIT_ACTIONS	UNCORRECTABLE
CURRENT_ROLE	INIT_ACTIONS	PRE_READ_ACTIONS	UNIT
CYCLE_TIME	INITIAL_VALUE	PRE_WRITE_ACTIONS	UNIT_ITEMS
DATA	INITIAL_VALUES	PREV	UNITS
DATA_ENTRY_ERROR	INLINE	PREV_COMPONENT	UNIVERSAL
DATA_ENTRY_WARNING	INPUT	PROCESS	unsigned
DATA_EXCHANGE	int	PROCESS_ERROR	UNSIGNED_INTEGER
DATA1	INTEGER	PROCESS_VALUE	VALIDITY
DATA2	INTERFACE	PROFIBUS_DP	VARIABLE
DATA3	INTERFACES	PROFIBUS_PA	VARIABLE_LIST
DATA4	ITEM_ARRAY	PROFINET	VARIABLE_LISTS
DATA5	ITEM_ARRAY_ITEMS	PROGRAM	VARIABLE_STATUS
DATA6	ITEM_ARRAYS	PROGRAMS	VARIABLES
DATA7	ITEMS	PROTOCOL	VECTORS
DATA8	KEY_POINTS	PURPOSE	VERTICAL
DATA9	LABEL	RDEFINE	VERTICAL_BAR
DATE	LARGE	READ	VIEW_MAX
DATE_AND_TIME	LAST	READ_ONLY	VIEW_MIN
DD_ITEM	LENGTH	READ_TIMEOUT	VISIBLE
DD_REVISION	LIKE	RECORD	VISIBLE
DD_STRING	LINE_COLOR	RECORDER	AVERTISSEMENT
DECLARATION	LINE_TYPE	RECORDS	WAVEFORM
default	LINEAR	REDEFINE	WAVEFORM_ITEMS
DEFAULT	LINK	REDEFINITIONS	WAVEFORMS
DEFAULT_VALUE	LIST	REDUNDANCY	while
DEFAULT_VALUES	LIST_ITEMS	REFRESH	WIDTH
DEFINITION	LISTS	REFRESH_ACTIONS	WINDOW
DELETE	LOAD_TO_APPLICATION	REFRESH_ITEMS	WRITE
DETAIL	LOAD_TO_DEVICE	REFRESHES	WRITE_AS_ONE
DETECT	LOCAL	RELATION_TYPE	WRITE_AS_ONE_ITEMS
DEVICE	LOCAL_DISPLAY	RELATIONS	WRITE_AS_ONES
DEVICE_REVISION	LOCAL_PARAM	REMOTEIO	WRITE_TIMEOUT
DEVICE_TYPE	LOCAL_PARAMETERS	REPLY	X_AXIS
DIAGNOSE	LOGARITHMIC	REQUEST	X_INCREMENT
DIAGNOSTIC	long	REQUIRED_INTERFACE	X_INITIAL
DIALOG	LOW_LIMIT	REQUIRED_RANGES	X_LARGE

Mot clé EDDL	Mot clé EDDL	Mot clé EDDL	Mot clé EDDL
DIGITAL_INPUT	LOW_LOW_LIMIT	RESPONSE_CODES	X_SMALL
DIGITAL_OUTPUT	MANUFACTURER	return	X_VALUES
DISCRETE_IN	MANUFACTURER_EXT	REVIEW	XX_LARGE
DISCRETE_INPUT	MAX_VALUE	ROLE	XX_SMALL
DISCRETE_OUT	MAX_VALUE1	ROOT	XY
DISCRETE_OUTPUT	MAX_VALUE2	ROWBREAK	Y_AXIS
DISPLAY_FORMAT	MAX_VALUE3	SCALING	Y_VALUES
DISPLAY_ITEMS	MAX_VALUE4	SCALING_FACTOR	YT
DOMAIN	MAX_VALUE5	SCAN	

A.5 Terminaux

Le texte suivant contient des terminaux autorisés de la structure lexicale EDDL.

```

DEFINE digit                = { 0-9 } .
    bin_digit               = { 0 1 } .
    non_zero_digit         = { 1-9 '-' } .
    oct_digit               = { 0-7 } .
    hex_digit               = { 0-9abcdefABCDEF } .
    letter                  = { a-zA-Z } .
    escapes                 = { '\"?afrty'\'} .
    ISOLatin1char           = - { " } .

/* Integer */
(0b|0B) bin_digit +        /* binary */
non_zero_digit digit *    /* decimal */
"0" oct_digit *           /* octal */
(0x|0X) hex_digit +       /* hexadecimal */

/* Real */
digit* "." digit+ ((E|e) {+|-}? digit+)?

/* String */
\" ISOLatin1char * \      /* ISO Latin-1 */
\" UTF8char * \          /* UTF-8 */

/* Character */
\' ISOLatin1char \'      /* ISO Latin-1 */
\' UTF8char \'           /* UTF-8 */

```

NOTE Une EDD doit utiliser un et un seul schéma de codage de caractère (c'est-à-dire, ISO Latin-1 ou UTF-8)

```

/* Identifier */
letter (letter|digit|_)*

```

A.6 Syntaxe EDDL formelle

A.6.1 Généralités

Cette annexe contient une syntaxe formelle du langage EDDL utilisant la forme de Backus-Naur.

S'il existe des éléments facultatifs, chaque élément est marqué avec /* M */ si l'élément est obligatoire ou /* O */ si l'élément est facultatif.

Chaque règle de forme de Backus-Naur est identifiée par un identifiant suivi par un deux-points.

A.6.2 Informations d'identification d'EDD

```
device_description:
  = identification definition_list
```

```
identification:
  = manufacturer ',' device_type ',' device_revision ','
    DD_revision
  = manufacturer ',' device_type ',' device_revision ','
    DD_revision ',' edd_version
  = manufacturer ',' device_type ',' device_revision ','
    DD_revision ',' edd_profile
  = manufacturer ',' device_type ',' device_revision ','
    DD_revision ',' manufacturer_ext
  = manufacturer ',' device_type ',' device_revision ','
    DD_revision ',' edd_version ',' edd_profile
  = manufacturer ',' device_type ',' device_revision ','
    DD_revision ',' edd_version ',' manufacturer_ext
  = manufacturer ',' device_type ',' device_revision ','
    DD_revision ',' edd_profile ',' manufacturer_ext
  = manufacturer ',' device_type ',' device_revision ','
    DD_revision ',' edd_version ',' edd_profile ','
    manufacturer_ext
```

```
manufacturer:
  = 'MANUFACTURER' Integer
  = 'MANUFACTURER' Identifier
```

```
device_type:
  = 'DEVICE_TYPE' Integer
  = 'DEVICE_TYPE' Identifier
```

```
device_revision:
  = 'DEVICE_REVISION' Integer
```

```
DD_revision:
  = 'DD_REVISION' Integer
  = 'EDD_REVISION' Integer
```

```
edd_version:
  = 'EDD_VERSION' Integer
```

```
edd_profile:
  = 'EDD_PROFILE' Integer
```

```
manufacturer_ext:
  = 'MANUFACTURER_EXT' string
```

```
definition_list:
  = definition
  = definition_list definition
```

```
definition:
  = item
  = imported_description
  = like
```

```
item:
  = axis
  = array
```

```
= block
= chart
= collection
= command
= component
= component_folder
= component_reference
= component_relation
= connection
= domain
= edit_display
= file
= graph
= grid
= list
= image
= interface
= item_array
= menu
= method
= program
= record
= refresh_relation
= response_codes_definition
= source
= template
= unit_relation
= variable
= variable_list
= waveform
= wao_relation
```

A.6.3 AXIS

```
axis:
= 'AXIS' Identifier '{' axis_attribute_list '}'
= 'AXIS' Identifier '{' '}'

axis_attribute_list:
= axis_attribute
= axis_attribute_list axis_attribute

axis_attribute:
= constant_unit /* 0 */
= help /* 0 */
= label /* 0 */
= axis_max_value /* 0 */
= axis_min_value /* 0 */
= axis_scaling /* 0 */

axis_min_value:
= minimum_value

axis_max_value:
= maximum_value

axis_scaling:
= 'SCALING' axis_scaling_specifier

axis_scaling_specifier:
= axis_scaling_keyword ';'
= 'IF' '(' expr ')' '{' axis_scaling_specifier '}'
= 'IF' '(' expr ')' '{' axis_scaling_specifier '}'
'ELSE' '{' axis_scaling_specifier '}'
```

```
= 'SELECT' '(' expr ')' '{' axis_scaling_selection_list '}'
```

```
axis_scaling_selection_list:
= axis_scaling_selection
= axis_scaling_selection_list axis_scaling_selection
```

```
axis_scaling_selection:
= 'CASE' expr ':' axis_scaling_specifier
= 'DEFAULT' ':' axis_scaling_specifier
```

```
axis_scaling_keyword:
= 'LINEAR'
= 'LOGARITHMIC'
```

A.6.4 BLOCK_A et BLOCK_B

```
block:
= 'BLOCK' Identifier '{' block_attribute_list '}'
```

```
block_attribute_list:
= block_attribute
= block_attribute_list block_attribute
```

```
block_attribute:
= block_a_characteristics /* M */
= label /* M */
= block_a_parameters /* M */
= block_a_axis_items /* O */
= block_a_chart_items /* O */
= block_a_collection_items /* O */
= block_a_edit_display_items /* O */
= block_a_file_items /* O */
= block_a_graph_items /* O */
= block_a_grid_items /* O */
= help /* O */
= block_a_image_items /* O */
= block_a_item_array_items /* O */
= block_a_list_items /* O */
= block_a_local_parameters /* O */
= block_a_menu_items /* O */
= block_a_method_items /* O */
= block_a_parameter_lists /* O */
= block_a_refresh_items /* O */
= block_a_source_items /* O */
= block_a_unit_items /* O */
= block_a_wao_items /* O */
= block_a_waveform_items /* O */
= block_a_charts /* O */
= block_a_lists /* O */
= block_a_graphs /* O */
= block_a_grids /* O */
= block_a_menus /* O */
= block_a_methods /* O */
= block_b_number /* M */
= block_b_type /* M */
```

```
block_a_characteristics:
= 'CHARACTERISTICS' record_reference ';' ;
```

```
block_a_parameters:
= 'PARAMETERS' '{' member_list '}'
```

```
member_list:
= member
```

```

= member_list member

block_a_axis_items:
= 'AXIS_ITEMS' '{' axis_items_specifier_list '}'

axis_items_specifier_list:
= axis_items_specifier
= axis_items_specifier_list ',' axis_items_specifier

axis_items_specifier:
= axis_reference

block_a_chart_items:
= 'CHART_ITEMS' '{' chart_items_specifier_list '}'

chart_items_specifier_list:
= chart_items_specifier
= chart_items_specifier_list ',' chart_items_specifier

chart_items_specifier:
= chart_reference

block_a_collection_items:
= 'COLLECTION_ITEMS' '{' collection_items_specifier_list '}'

collection_items_specifier_list:
= collection_items_specifier
= collection_items_specifier_list ',' collection_items_specifier

collection_items_specifier:
= collection_reference

block_a_edit_display_items:
= 'EDIT_DISPLAY_ITEMS' '{' edit_display_items_specifier_list '}'

edit_display_items_specifier_list:
= edit_display_items_specifier
= edit_display_items_specifier_list ',' edit_display_items_specifier

edit_display_items_specifier:
= edit_display_reference

block_a_file_items:
= 'FILE_ITEMS' '{' file_items_specifier_list '}'

file_items_specifier_list:
= file_items_specifier
= file_items_specifier_list ',' file_items_specifier

file_items_specifier:
= file_reference

block_a_graph_items:
= 'GRAPH_ITEMS' '{' graph_items_specifier_list '}'

graph_items_specifier_list:
= graph_items_specifier
= graph_items_specifier_list ',' graph_items_specifier

graph_items_specifier:
= graph_reference

block_a_grid_items:
= 'GRID_ITEMS' '{' grid_items_specifier_list '}'

```

```

grid_items_specifier_list:
  = grid_items_specifier
  = grid_items_specifier_list ',' grid_items_specifier

grid_items_specifier:
  = grid_reference

block_a_image_items:
  = 'IMAGE_ITEMS' '{' image_items_specifier_list '}'

image_items_specifier_list:
  = image_items_specifier
  = image_items_specifier_list ',' image_items_specifier

image_items_specifier:
  = image_reference

block_a_item_array_items:
  = 'ITEM_ARRAY_ITEMS' '{' item_array_items_specifier_list '}'

item_array_items_specifier_list:
  = item_array_items_specifier
  = item_array_items_specifier_list ',' item_array_items_specifier

item_array_items_specifier:
  = item_array_reference

block_a_list_items:
  = 'LIST_ITEMS' '{' list_items_specifier_list '}'

list_items_specifier_list:
  = list_items_specifier
  = list_items_specifier_list ',' list_items_specifier

list_items_specifier:
  = list_reference

block_a_local_parameters:
  = 'LOCAL_PARAMETERS' '{' member_list '}'

block_a_menu_items:
  = 'MENU_ITEMS' '{' block_a_menu_items_specifier_list '}'

block_a_menu_items_specifier_list:
  = block_a_menu_items_specifier
  = block_a_menu_items_specifier_list ',' block_a_menu_items_specifier

block_a_menu_items_specifier:
  = menu_reference

block_a_method_items:
  = 'METHOD_ITEMS' '{' method_items_specifier_list '}'

method_items_specifier_list:
  = method_items_specifier
  = method_items_specifier_list ',' method_items_specifier

method_items_specifier:
  = method_reference

block_a_parameter_lists:
  = 'PARAMETER_LISTS' '{' member_list '}'

```

```

block_a_refresh_items:
    = 'REFRESH_ITEMS' '{ refresh_items_specifier_list }'

refresh_items_specifier_list:
    = refresh_items_specifier
    = refresh_items_specifier_list ',' refresh_items_specifier

refresh_items_specifier:
    = refresh_reference

block_a_source_items:
    = 'SOURCE_ITEMS' '{ source_items_specifier_list }'

source_items_specifier_list:
    = source_items_specifier
    = source_items_specifier_list ',' source_items_specifier

source_items_specifier:
    = source_reference

block_a_unit_items:
    = 'UNIT_ITEMS' '{ unit_items_specifier_list }'

unit_items_specifier_list:
    = unit_items_specifier
    = unit_items_specifier_list ',' unit_items_specifier

unit_items_specifier:
    = unit_reference

block_a_wao_items:
    = 'WRITE_AS_ONE_ITEMS' '{ wao_items_specifier_list }'

wao_items_specifier_list:
    = wao_items_specifier
    = wao_items_specifier_list ',' wao_items_specifier

wao_items_specifier:
    = wao_reference

block_a_waveform_items:
    = 'WAVEFORM_ITEMS' '{ waveform_items_specifier_list }'

waveform_items_specifier_list:
    = waveform_items_specifier
    = waveform_items_specifier_list ',' waveform_items_specifier

waveform_items_specifier:
    = waveform_reference

block_a_charts:
    = 'CHARTS' '{ member_list }'

block_a_lists:
    = 'LISTS' '{ member_list }'

block_a_graphs:
    = 'GRAPHS' '{ member_list }'

block_a_grids:
    = 'GRIDS' '{ member_list }'

```



```

block_a_menus:
    = 'MENUS' '{' member_list '}'

block_a_methods:
    = 'METHODS' '{' member_list '}'

block_b_number:
    = 'NUMBER' expr ';'

block_b_type:
    = 'TYPE' 'PHYSICAL' ';'
    = 'TYPE' 'TRANSDUCER' ';'
    = 'TYPE' 'FUNCTION' ';'

```

A.6.5 CHART

```

chart:
    = 'CHART' Identifier '{' chart_attribute_list '}'

chart_attribute_list:
    = chart_attribute
    = chart_attribute_list chart_attribute

chart_attribute:
    = members                /* M */
    = chart_cycle_time       /* O */
    = help                   /* O */
    = height                 /* O */
    = label                  /* O */
    = chart_length           /* O */
    = chart_type             /* O */
    = validity               /* O */
    = width                  /* O */

chart_cycle_time:
    = 'CYCLE_TIME' expr_specifier

chart_length:
    = 'LENGTH' expr_specifier

chart_type:
    = 'TYPE' chart_type_specifier

chart_type_specifier:
    = chart_type_keyword ';'
    = 'IF' '(' expr ')' '{' chart_type_specifier '}'
    = 'IF' '(' expr ')' '{' chart_type_specifier '}'
    = 'ELSE' '{' chart_type_specifier '}'
    = 'SELECT' '(' expr ')' '{' chart_type_selection_list '}'

chart_type_selection_list:
    = chart_type_selection
    = chart_type_selection_list chart_type_selection

chart_type_selection:
    = 'CASE' expr ':' chart_type_specifier
    = 'DEFAULT' ':' chart_type_specifier

chart_type_keyword:
    = 'GAUGE'
    = 'HORIZONTAL_BAR'
    = 'SCOPE'
    = 'STRIP'

```

```
= 'SWEEP'
= 'VERTICAL_BAR'
```

A.6.6 COLLECTION

```
collection:
= 'COLLECTION' Identifier '{' collection_attribute_list '}'
= 'COLLECTION' 'OF' item_type Identifier '{'
  collection_attribute_list '}'
```

```
item_type:
= 'ARRAY'
= 'AXIS'
= 'BLOCK'
= 'CHART'
= 'COLLECTION' 'OF' item_type
= 'COMMAND'
= 'COMPONENT'
= 'COMPONENT_FOLDER'
= 'COMPONENT_REFERENCE'
= 'COMPONENT_RELATION'
= 'CONNECTION'
= 'DOMAIN'
= 'EDIT_DISPLAY'
= 'FILE'
= 'GRAPH'
= 'GRID'
= 'IMAGE'
= 'INTERFACE'
= 'ITEM_ARRAY' 'OF' item_type
= 'LIST'
= 'MENU'
= 'METHOD'
= 'PROGRAM'
= 'RECORD'
= 'REFRESH'
= 'RESPONSE_CODES'
= 'SOURCE'
= 'TEMPLATE'
= 'UNIT'
= 'VARIABLE'
= 'VARIABLE_LIST'
= 'WAVEFORM'
= 'WRITE_AS_ONE'
```

```
collection_attribute_list:
= collection_attribute
= collection_attribute_list collection_attribute
```

```
collection_attribute:
= members      /* M */
= help         /* O */
= label        /* O */
= validity     /* O */
```

A.6.7 COMMAND

```
command:
= 'COMMAND' Identifier '{' command_attribute_list '}'
= 'COMMAND' Identifier '{' '}'
```

```
command_attribute_list:
= command_attribute
= command_attribute_list command_attribute
```

```

command_attribute:
= command_operation          /* M */
= command_transaction        /* M */
= command_index              /* O */
= command_block              /* O */
= command_number             /* O */
= command_slot               /* O */
= command_sub_slot           /* O */
= command_connection         /* O */
= command_header             /* O */
= command_module             /* O */
= response_codes             /* O */

command_operation:
= 'OPERATION' operation ';'

operation:
= 'READ'
= 'WRITE'
= 'COMMAND'
= 'DATA_EXCHANGE'
= string

command_transaction:
= 'TRANSACTION' '{' transaction_attribute_list '}'
= 'TRANSACTION' Integer '{' transaction_attribute_list '}'

transaction_attribute_list:
= transaction_attribute
= transaction_attribute_list transaction_attribute

transaction_attribute:
= request                      /* M */
= reply                        /* M */
= 'RESPONSE_CODES' '(' response_code_reference ')' /* O */

request:
= 'REQUEST' '{' data_items_specifier_list '}'
= 'REQUEST' '{' '}'

reply:
= 'REPLY' '{' data_items_specifier_list '}'
= 'REPLY' '{' '}'

data_items_specifier_list:
= data_items_specifier
= data_items_specifier_list ',' data_items_specifier

data_items_specifier:
= Integer
= variable_reference
= variable_reference '<' Integer '>'
= variable_reference '(' data_items_qualifier_list ')'
= variable_reference '<' Integer '>' '(' data_items_qualifier_list ')'
= array_reference
= array_reference '(' data_items_qualifier_list ')'
= collection_reference
= collection_reference '(' data_items_qualifier_list ')'
= item_array_reference
= item_array_reference '(' data_items_qualifier_list ')'
= list_reference
= list_reference '(' data_items_qualifier_list ')'

```

```

data_items_qualifier_list:
  = data_items_qualifier
  = data_items_qualifier_list ',' data_items_qualifier

data_items_qualifier:
  = 'INDEX'
  = 'INFO'

command_index:
  = 'INDEX' index_specifier

index_specifier:
  = index_items ';'
  = 'IF' '(' expr ')' '{' index_specifier '}'
  = 'IF' '(' expr ')' '{' index_specifier '}'
  'ELSE' '{' index_specifier '}'
  = 'SELECT' '(' expr ')' '{' index_selection_list '}'

index_items:
  = expr
  = variable_reference

index_selection_list:
  = index_selection
  = index_selection_list index_selection

index_selection:
  = 'CASE' expr ':' index_specifier
  = 'DEFAULT' ':' index_specifier

command_block:
  = 'BLOCK' block_specifier

block_specifier:
  = block_b_reference ';'
  = 'IF' '(' expr ')' '{' block_specifier '}'
  = 'IF' '(' expr ')' '{' block_specifier '}'
  'ELSE' '{' block_specifier '}'
  = 'SELECT' '(' expr ')' '{' block_selection_list '}'

block_selection_list:
  = block_selection
  = block_selection_list block_selection

block_selection:
  = 'CASE' expr ':' block_specifier
  = 'DEFAULT' ':' block_specifier

command_number:
  = 'NUMBER' number_specifier

number_specifier:
  = number_items ';'
  = 'IF' '(' expr ')' '{' number_specifier '}'
  = 'IF' '(' expr ')' '{' number_specifier '}'
  'ELSE' '{' number_specifier '}'
  = 'SELECT' '(' expr ')' '{' number_selection_list '}'

number_items:
  = Integer
  = variable_reference

number_selection_list:

```

```

= number_selection
= number_selection_list number_selection

number_selection:
= 'CASE' expr ':' number_specifier
= 'DEFAULT' ':' number_specifier

command_slot:
= 'SLOT' slot_specifier

slot_specifier:
= slot_items ';'
= 'IF' '(' expr ')' '{' slot_specifier '}'
= 'IF' '(' expr ')' '{' slot_specifier '}'
  'ELSE' '{' slot_specifier '}'
= 'SELECT' '(' expr ')' '{' slot_selection_list '}'

slot_items:
= expr
= variable_reference

slot_selection_list:
= slot_selection
= slot_selection_list slot_selection

slot_selection:
= 'CASE' expr ':' slot_specifier
= 'DEFAULT' ':' slot_specifier

command_sub_slot:
= 'SUB_SLOT' slot_specifier

command_connection:
= 'CONNECTION' connection_specifier

connection_specifier:
= connection_reference ';'
= 'IF' '(' expr ')' '{' connection_specifier '}'
= 'IF' '(' expr ')' '{' connection_specifier '}'
  'ELSE' '{' connection_specifier '}'
= 'SELECT' '(' expr ')' '{' connection_selection_list '}'

connection_selection_list:
= connection_selection
= connection_selection_list connection_selection

connection_selection:
= 'CASE' expr ':' connection_specifier
= 'DEFAULT' ':' connection_specifier

command_header:
= 'HEADER' header_specifier

header_specifier:
= string ';'
= 'IF' '(' expr ')' '{' header_specifier '}'
= 'IF' '(' expr ')' '{' header_specifier '}'
  'ELSE' '{' header_specifier '}'
= 'SELECT' '(' expr ')' '{' header_selection_list '}'

header_selection_list:
= header_selection
= header_selection_list header_selection

```

```

header_selection:
  = 'CASE' expr ':' header_specifier
  = 'DEFAULT' ':' header_specifier

command_module:
  = 'MODULE' module_specifier

module_specifier:
  = Identifier ';'
  = 'IF' '(' expr ')' '{' module_specifier '}'
  = 'IF' '(' expr ')' '{' module_specifier '}'
    'ELSE' '{' module_specifier '}'
  = 'SELECT' '(' expr ')' '{' module_selection_list '}'

module_selection_list:
  = module_selection
  = module_selection_list module_selection

module_selection:
  = 'CASE' expr ':' module_specifier
  = 'DEFAULT' ':' module_specifier

```

A.6.8 COMPONENT

```

component:
  = 'COMPONENT' Identifier '{' component_attribute_list '}'

component_attribute_list:
  = component_attribute
  = component_attribute_list component_attribute

component_attribute:
  = label /* M */
  = can_delete /* O */
  = check_configuration /* O */
  = classification /* O */
  = component_parent /* O */
  = component_path /* O */
  = component_relations /* O */
  = declaration /* O */
  = detect_method /* O */
  = edd /* O */
  = help /* O */
  = initial_values /* O */
  = redundancy /* O */
  = protocol /* O */
  = scan_method /* O */
  = scan_list /* O */
  = supplied_interface /* O */

can_delete:
  = 'CAN_DELETE' boolean_specifier ';'

check_configuration:
  = 'CHECK_CONFIGURATION' expr_specifier

classification:
  = 'CLASSIFICATION' classification_specifier ';'

classification_specifier:
  = 'ACTUATOR'
  = 'ACTUATOR_ELECTRO_PNEUMATIC'

```

```

= 'ACTUATOR_ELECTRIC'
= 'ACTUATOR_HYDRAULIC'
= 'CONVERTER'
= 'CONTROLLER'
= 'DISCRETE_IN'
= 'DISCRETE_OUT'
= 'ELECTRICAL_DISTRIBUTION'
= 'ELECTRICAL_DISTRIBUTION_POWER_MONITORING'
= 'FREQUENCY_CONVERTER'
= 'INDICATOR'
= 'NETWORK_COMPONENT'
= 'NETWORK_COMPONENT_WIRELESSADAPTER'
= 'REMOTEIO'
= 'RECORDER'
= 'SENSOR'
= 'SENSOR_ACOUSTIC'
= 'SENSOR_ANALYTIC'
= 'SENSOR_ANALYTIC_GAS'
= 'SENSOR_ANALYTIC_LIQUID'
= 'SENSOR_CONCENTRATION'
= 'SENSOR_DENSITY'
= 'SENSOR_DENSITY_RADIOMETRIC'
= 'SENSOR_FLOW'
= 'SENSOR_FLOW_CORIOLIS'
= 'SENSOR_FLOW_ELECTRO_MAGNETIC'
= 'SENSOR_FLOW_MECHANICAL'
= 'SENSOR_FLOW_RADIOMETRIC'
= 'SENSOR_FLOW_THERMAL'
= 'SENSOR_FLOW_ULTRASONIC'
= 'SENSOR_FLOW_VORTEX_COUNTER'
= 'SENSOR_LEVEL'
= 'SENSOR_LEVEL_BUOYANCY'
= 'SENSOR_LEVEL_CAPACITIVE'
= 'SENSOR_LEVEL_ECHO'
= 'SENSOR_LEVEL_HYDROSTATIC'
= 'SENSOR_LEVEL_RADIOMETRIC'
= 'SENSOR_MULTIVARIABLE'
= 'SENSOR_POSITION'
= 'SENSOR_PRESSURE'
= 'SENSOR_TEMPERATURE'
= 'SEMICONDUCTOR'
= 'SWITCHGEAR'
= 'UNIVERSAL'
= string

component_path:
= 'COMPONENT_PATH' string ';'

component_parent:
= 'COMPONENT_PARENT' reference ';'

component_relations:
= 'COMPONENT_RELATIONS' '{' component_specifier_list '}'

component_specifier_list:
= component_specifier
= component_specifier_list ',' component_specifier

component_specifier:
= Identifier
= 'IF' '(' expr ')' '{' component_specifier_list '}'
= 'IF' '(' expr ')' '{' component_specifier_list '}'
'ELSE' '{' component_specifier_list '}'

```

```

= 'SELECT' '(' expr ')' '{' component_selection_list '}'

component_selection_list:
= component_selection
= component_selection_list component_selection

component_selection:
= 'CASE' expr ':' component_specifier_list
= 'DEFAULT' ':' component_specifier_list

declaration:
= 'DECLARATION' '{' item_list '}'

item_list:
= item
= item_list item

detect_method:
= 'DETECT' method_reference ';'

edd:
= 'EDD' string ';'

protocol:
= 'PROTOCOL' protocol_specifier ';'

protocol_specifier:
= 'PROFIBUS_DP'
= 'PROFIBUS_PA'
= 'PROFINET'
= 'HART'
= 'FF'
= string

redundancy:
= 'REDUNDANCY' expr_specifier

initial_values:
= 'INITIAL_VALUES' '{' initial_value_list '}'

initial_value_list:
= initial_value_specifier
= initial_value_list initial_value_specifier

initial_value_specifier:
= reference_without_call '=' expr_specifier

scan_method:
= 'SCAN' method_reference ';'

scan_list:
= 'SCAN_LIST' reference_without_call ';'

supplied_interface:
= 'SUPPLIED_INTERFACE' '{' interface_list '}'

```

A.6.9 COMPONENT_FOLDER

```

component_folder:
= 'COMPONENT_FOLDER' Identifier '{' component_folder_attribute_list '}'

component_folder_attribute_list:
= component_folder_attribute
= component_folder_attribute_list component_folder_attribute

```



```

component_folder_attribute:
= label                /* M */
= classification        /* O */
= component_parent      /* O */
= component_path        /* O */
= help                 /* O */
= protocol              /* O */

```

A.6.10 COMPONENT_REFERENCE

```

component_reference:
= 'COMPONENT_REFERENCE' Identifier '{'
  component_reference_attribute_list '}'

```

```

component_reference_attribute_list:
= component_reference_attribute
= component_reference_attribute_list component_reference_attribute

```

```

component_reference_attribute:
= classification        /* O */
= component_parent      /* O */
= component_path        /* O */
= device_revision_attribute /* O */
= device_type_attribute /* O */
= manufacturer_attribute /* O */
= protocol              /* O */

```

```

device_revision_attribute:
= device_revision ';'

```

```

device_type_attribute:
= device_type ';'

```

```

manufacturer_attribute:
= manufacturer ';'

```

A.6.11 COMPONENT_RELATION

```

component_relation:
= 'COMPONENT_RELATION' Identifier '{'
  component_relation_attribute_list '}'

```

```

component_relation_attribute_list:
= component_relation_attribute
= component_relation_attribute_list component_relation_attribute

```

```

component_relation_attribute:
= components            /* M */
= label                 /* M */
= relation_type         /* M */
= addressing            /* O */
= help                 /* O */
= maximum_number        /* O */
= minimum_number        /* O */
= required_interface    /* O */
= supplied_interface    /* O */

```

```

components:
= 'COMPONENTS' '{' component_relation_reference_list '}'

```

```

component_relation_reference_list:
= component_relation_reference
= component_relation_reference_list ','
  component_relation_reference

```

```

component_relation_reference:
  = component_relation_reference_specifier
  = 'IF' '(' expr ')' '{' component_relation_reference_list '}'
  = 'IF' '(' expr ')' '{' component_relation_reference_list '}'
    'ELSE' '{' component_relation_reference_list '}'
  = 'SELECT' '(' expr ')' '{' component_relation_selection_list '}'

component_relation_selection_list:
  = component_relation_selection
  = component_relation_selection_list component_relation_selection

component_relation_selection:
  = 'CASE' expr ':' component_relation_reference_list
  = 'DEFAULT' ':' component_relation_reference_list

component_relation_reference_specifier:
  = Identifier
  = Identifier '{' component_relation_arg_list '}'

component_relation_arg_list:
  = component_relation_arg
  = component_relation_arg_list component_relation_arg

component_relation_arg:
  = minimum_number
  = maximum_number
  = auto_create
  = filter
  = required_ranges

auto_create:
  = 'AUTO_CREATE' expr_specifier

filter:
  = 'FILTER' '(' expr ')'

required_ranges:
  = 'REQUIRED_RANGES' '{' required_ranges_list '}'

required_ranges_list:
  = required_range_specifier
  = required_ranges_list ';' required_range_specifier

required_range_specifier:
  = variable_reference '{' variable_range_arg_list '}'

variable_range_arg_list:
  = variable_range_arg
  = variable_range_arg_list variable_range_arg

variable_range_arg:
  = maximum_value
  = maximum_value_n
  = minimum_value
  = minimum_value_n

relation_type:
  = 'RELATION_TYPE' relation_type_specifier ';'

relation_type_specifier:
  = 'PARENT_COMPONENT'

```

```

= 'CHILD_COMPONENT'
= 'SIBLING_COMPONENT'
= 'NEXT_COMPONENT'
= 'PREV_COMPONENT'

```

```

addressing:
= 'ADDRESSING' '{' addressing_list '}'

```

```

addressing_list:
= variable_reference
= addressing_list ',' variable_reference

```

```

maximum_number:
= 'MAXIMUM_NUMBER' expr_specifier

```

```

minimum_number:
= 'MINIMUM_NUMBER' expr_specifier

```

```

required_interface:
= 'REQUIRED_INTERFACE' '{' interface_list '}'

```

```

interface_list:
= interface_reference
= interface_list ',' interface_reference

```

```

interface_reference:
= Identifier

```

A.6.12 CONNECTION

```

connection:
= 'CONNECTION' Identifier '{' connection_attribute_list '}'

```

```

connection_attribute_list:
= connection_attribute
= connection_attribute_list connection_attribute

```

```

connection_attribute:
= appinstance /* M */

```

```

appinstance:
= 'APPINSTANCE' Integer ';'

```

A.6.13 DOMAIN

```

domain:
= 'DOMAIN' Identifier '{' domain_attribute_list '}'
= 'DOMAIN' Identifier '{' '}'

```

```

domain_attribute_list:
= domain_attribute
= domain_attribute_list domain_attribute

```

```

domain_attribute:
= handling /* O */
= response_codes /* O */

```

A.6.14 EDIT_DISPLAY

```

edit_display:
= 'EDIT_DISPLAY' Identifier '{' edit_display_attribute_list '}'

```

```

edit_display_attribute_list:
= edit_display_attribute
= edit_display_attribute_list edit_display_attribute

```

```

edit_display_attribute:
= edit_items          /* M */
= label              /* M */
= display_items       /* O */
= post_edit_actions   /* O */
= pre_edit_actions    /* O */

edit_items:
= 'EDIT_ITEMS' '{' edit_items_specifier_list '}'

edit_items_specifier_list:
= edit_items_specifier
= edit_items_specifier_list edit_items_specifier

edit_items_specifier:
= edit_item_list
= 'IF' '(' expr ')' '{' edit_items_specifier_list '}'
= 'IF' '(' expr ')' '{' edit_items_specifier_list '}'
  'ELSE' '{' edit_items_specifier_list '}'
= 'SELECT' '(' expr ')' '{' edit_items_selection_list '}'

edit_items_selection_list:
= edit_items_selection
= edit_items_selection_list edit_items_selection

edit_items_selection:
= 'CASE' expr ':' edit_items_specifier_list
= 'DEFAULT' ':' edit_items_specifier_list

edit_item_list:
= edit_item
= edit_item_list ',' edit_item

edit_item:
= reference_without_call

display_items:
= 'DISPLAY_ITEMS' '{' display_items_specifier_list '}'

display_items_specifier_list:
= display_items_specifier
= display_items_specifier_list display_items_specifier

display_items_specifier:
= display_item_list
= 'IF' '(' expr ')' '{' display_items_specifier_list '}'
= 'IF' '(' expr ')' '{' display_items_specifier_list '}'
  'ELSE' '{' display_items_specifier_list '}'
= 'SELECT' '(' expr ')' '{' display_items_selection_list '}'

display_items_selection_list:
= display_items_selection
= display_items_selection_list display_items_selection

display_items_selection:
= 'CASE' expr ':' display_items_specifier_list
= 'DEFAULT' ':' display_items_specifier_list

display_item_list:
= display_item
= display_item_list ',' display_item

display_item:

```

= variable_reference

A.6.15 FILE

```
file:
= 'FILE' Identifier '{' file_attribute_list '}'
```

```
file_attribute_list:
= file_attribute
= file_attribute_list file_attribute
```

```
file_attribute:
= members      /* M */
= label        /* O */
= help         /* O */
```

A.6.16 GRAPH

```
graph:
= 'GRAPH' Identifier '{' graph_attribute_list '}'
```

```
graph_attribute_list:
= graph_attribute
= graph_attribute_list graph_attribute
```

```
graph_attribute:
= members      /* M */
= label        /* O */
= help         /* O */
= height       /* O */
= graph_cycle_time /* O */
= validity     /* O */
= width        /* O */
= graph_x_axis /* O */
```

```
graph_cycle_time:
= 'CYCLE_TIME' expr_specifier
```

```
graph_x_axis:
= 'X_AXIS' axis_specifier
```

A.6.17 GRID

```
grid:
= 'GRID' Identifier '{' grid_attribute_list '}'
```

```
grid_attribute_list:
= grid_attribute
= grid_attribute_list grid_attribute
```

```
grid_attribute:
= grid_vectors      /* M */
= handling           /* O */
= height             /* O */
= help               /* O */
= label              /* O */
= grid_orientation   /* O */
= validity           /* O */
= width              /* O */
```

```
grid_vectors:
= 'VECTORS' '{' vector_specifier_list '}'
```

```
vector_specifier_list:
= vector_specifier
= vector_specifier_list ',' vector_specifier
```

```
vector_specifier:
  = vector
  = 'IF' '(' expr ')' '{' vector_specifier_list '}'
  = 'IF' '(' expr ')' '{' vector_specifier_list '}'
    'ELSE' '{' vector_specifier_list '}'
  = 'SELECT' '(' expr ')' '{' vector_selection_list '}'
```

```
vector_selection_list:
  = vector_selection
  = vector_selection_list vector_selection
```

```
vector_selection:
  = 'CASE' expr ':' vector_specifier_list
  = 'DEFAULT' ':' vector_specifier_list
```

```
vector:
  = '{' string ',' vector_values_list '}'
```

```
vector_values_list:
  = vector_values
  = vector_values_list ',' vector_values
```

```
vector_values:
  = reference_without_call
  = Integer
  = Real
  = string
```

```
grid_orientation:
  = 'ORIENTATION' orientation_specifier ';'

```

```
orientation_specifier:
  = 'HORIZONTAL'
  = 'VERTICAL'
```

A.6.18 IMAGE

```
image:
  = 'IMAGE' Identifier '{' image_attribute_list '}'
```

```
image_attribute_list:
  = image_attribute
  = image_attribute_list image_attribute
```

```
image_attribute:
  = image_path          /* M */
  = label                /* O */
  = help                 /* O */
  = image_link           /* O */
  = validity             /* O */
```

```
image_path:
  = 'PATH' file_name_specifier
```

```
file_name_specifier:
  = string_literal ';'
  = 'IF' '(' expr ')' '{' file_name_specifier '}'
  = 'IF' '(' expr ')' '{' file_name_specifier '}'
    'ELSE' '{' file_name_specifier '}'
  = 'SELECT' '(' expr ')' '{' file_name_selection_list '}'
```

```
file_name_selection_list:
  = file_name_selection
```

```

= file_name_selection_list file_name_selection

file_name_selection:
= 'CASE' expr ':' file_name_specifier
= 'DEFAULT' ':' file_name_specifier

image_link:
= 'LINK' link_specifier

link_specifier:
= reference_without_call ';'
= 'IF' '(' expr ')' '{' link_specifier '}'
= 'IF' '(' expr ')' '{' link_specifier '}'
  'ELSE' '{' link_specifier '}'
= 'SELECT' '(' expr ')' '{' link_selection_list '}'

link_selection_list:
= link_selection
= link_selection_list link_selection

link_selection:
= 'CASE' expr ':' link_specifier
= 'DEFAULT' ':' link_specifier

```

A.6.19 INTERFACE

```

interface:
= 'INTERFACE' Identifier '{' interface_attribute_list '}'

interface_attribute_list:
= interface_attribute
= interface_attribute_list interface_attribute

interface_attribute:
= declaration /* M */
= label /* M */
= help /* O */

```

A.6.20 LIST

```

list:
= 'LIST' Identifier '{' list_attribute_list '}'

list_attribute_list:
= list_attribute
= list_attribute_list list_attribute

list_attribute:
= list_type /* M */
= list_capacity /* O */
= list_count /* O */
= help /* O */
= label /* O */
= validity /* O */

list_type:
= 'TYPE' list_type_specifier ';'

list_type_specifier:
= reference_without_call

list_capacity:
= 'CAPACITY' Integer ';'

list_count:

```

```
= 'COUNT' expr_specifier
```

A.6.21 IMPORT

```
imported_description:
```

```
= 'IMPORT' identification '{' imports '}'
= 'IMPORT' identification '{' imports redefinitions '}'
= 'IMPORT' '"' filename '"'
= 'IMPORT' '<' filename '>'
```

```
imports:
```

```
= 'EVERYTHING' ';'
= item_import_list
```

```
item_import_list:
```

```
= item_import
= item_import_list item_import
```

```
item_import:
```

```
= item_import_by_name ';'
= item_import_by_type ';'

```

```
item_import_by_name:
```

```
= item_type Identifier
= Identifier
```

```
item_import_by_type:
```

```
= import_item_type
= item_import_by_type '&' import_item_type
```

```
import_item_type:
```

```
= 'ARRAYS'
= 'AXES'
= 'BLOCKS'
= 'CHARTS'
= 'COLLECTIONS'
= 'COMMANDS'
= 'COMPONENTS'
= 'COMPONENT_FOLDERS'
= 'COMPONENT_REFERENCES'
= 'COMPONENT_RELATIONS'
= 'CONNECTIONS'
= 'DOMAINS'
= 'EDIT_DISPLAYS'
= 'FILES'
= 'GRAPHS'
= 'GRIDS'
= 'IMAGES'
= 'INTERFACES'
= 'ITEM_ARRAYS'
= 'LISTS'
= 'MENUS'
= 'METHODS'
= 'PROGRAMS'
= 'RECORDS'
= 'REFRESHES'
= 'RELATIONS'
= 'RESPONSE_CODES'
= 'SOURCES'
= 'TEMPLATES'
= 'UNITS'
= 'VARIABLES'
= 'VARIABLE_LISTS'
= 'WRITE_AS_ONES'
```



```

= 'WAVEFORMS'

redefinitions:
= 'REDEFINITIONS' '{' redefinition_list '}'

redefinition_list:
= redefinition
= redefinition_list redefinition

redefinition:
= array_redefinition
= axis_redefinition
= block_redefinition
= chart_redefinition
= collection_redefinition
= command_redefinition
= component_redefinition
= component_folder_redefinition
= component_reference_redefinition
= component_relation_redefinition
= connection_redefinition
= domain_redefinition
= edit_display_redefinition
= file_redefinition
= graph_redefinition
= grid_redefinition
= image_redefinition
= interface_redefinition
= item_array_redefinition
= list_redefinition
= menu_redefinition
= method_redefinition
= program_redefinition
= record_redefinition
= refresh_relation_redefinition
= response_codes_definition_redefinition
= source_redefinition
= template_redefinition
= unit_relation_redefinition
= variable_redefinition
= variable_list_redefinition
= wao_relation_redefinition
= waveform_redefinition

filename:
= string_literal

```

A.6.22 LIKE

```

like:
= Identifier 'LIKE' Identifier
  '{' attribute_redefinition_list '}'
= Identifier 'LIKE' 'ARRAY' Identifier
  '{' array_attribute_redefinition_list '}'
= Identifier 'LIKE' 'AXIS' Identifier
  '{' axis_attribute_redefinition_list '}'
= Identifier 'LIKE' 'BLOCK' Identifier
  '{' block_attribute_redefinition_list '}'
= Identifier 'LIKE' 'CHART' Identifier
  '{' chart_attribute_redefinition_list '}'
= Identifier 'LIKE' 'COLLECTION' 'OF' item_type Identifier
  '{' collection_attribute_redefinition_list '}'
= Identifier 'LIKE' 'COMMAND' Identifier

```

```
'{' command_attribute_redefinition_list '}'
= Identifier 'LIKE' 'COMPONENT' Identifier
  {' component_attribute_redefinition_list '}'
= Identifier 'LIKE' 'COMPONENT_FOLDER' Identifier
  {' component_folder_attribute_redefinition_list '}'
= Identifier 'LIKE' 'COMPONENT_REFERENCE' Identifier
  {' component_reference_attribute_redefinition_list '}'
= Identifier 'LIKE' 'COMPONENT_RELATION' Identifier
  {' component_relation_attribute_redefinition_list '}'
= Identifier 'LIKE' 'CONNECTION' Identifier
  {' connection_attribute_redefinition_list '}'
= Identifier 'LIKE' 'DOMAIN' Identifier
  {' domain_attribute_redefinition_list '}'
= Identifier 'LIKE' 'EDIT_DISPLAY' Identifier
  {' edit_display_attribute_redefinition_list '}'
= Identifier 'LIKE' 'FILE' Identifier
  {' file_attribute_redefinition_list '}'
= Identifier 'LIKE' 'GRAPH' Identifier
  {' graph_attribute_redefinition_list '}'
= Identifier 'LIKE' 'GRID' Identifier
  {' grid_attribute_redefinition_list '}'
= Identifier 'LIKE' 'IMAGE' Identifier
  {' image_attribute_redefinition_list '}'
= Identifier 'LIKE' 'INTERFACE' Identifier
  {' interface_attribute_redefinition_list '}'
= Identifier 'LIKE' 'ITEM_ARRAY' 'OF' item_type
  Identifier
  {' item_array_attribute_redefinition_list '}'
= Identifier 'LIKE' 'LIST' Identifier
  {' list_attribute_redefinition_list '}'
= Identifier 'LIKE' 'MENU' Identifier
  {' menu_attribute_redefinition_list '}'
= Identifier 'LIKE' 'METHOD' Identifier
  {' method_attribute_redefinition_list '}'
= Identifier 'LIKE' 'PROGRAM' Identifier
  {' program_attribute_redefinition_list '}'
= Identifier 'LIKE' 'RECORD' Identifier
  {' record_attribute_redefinition_list '}'
= Identifier 'LIKE' 'REFRESH' Identifier
  {' refresh_relation_left ':' refresh_relation_right '}'
= Identifier 'LIKE' 'RESPONSE_CODES' Identifier
  {' response_code_redefinition_list '}'
= Identifier 'LIKE' 'SOURCE' Identifier
  {' source_attribute_redefinition_list '}'
= Identifier 'LIKE' 'TEMPLATE' Identifier
  {' template_attribute_redefinition_list '}'
= Identifier 'LIKE' 'UNIT' Identifier
  {' unit_relation_left ':' unit_relation_right '}'
= Identifier 'LIKE' 'VARIABLE' Identifier
  {' variable_attribute_redefinition_list '}'
= Identifier 'LIKE' 'VARIABLE_LIST' Identifier
  {' variable_list_attribute_redefinition_list '}'
= Identifier 'LIKE' 'WAVEFORM' Identifier
  {' waveform_attribute_redefinition_list '}'
= Identifier 'LIKE' 'WRITE_AS_ONE' Identifier
  {' wao_specifier '}'
```

```
attribute_redefinition_list:
= attribute_redefinition
= attribute_redefinition_list attribute_redefinition
```

```
attribute_redefinition:
= array_attribute_redefinition
```

```

= axis_attribute_redefinition
= block_attribute_redefinition
= chart_attribute_redefinition
= collection_attribute_redefinition
= command_attribute_redefinition
= component_attribute_redefinition
= component_folder_attribute_redefinition
= component_reference_attribute_redefinition
= component_relation_attribute_redefinition
= connection_attribute_redefinition
= domain_attribute_redefinition
= edit_display_attribute_redefinition
= file_attribute_redefinition
= graph_attribute_redefinition
= grid_attribute_redefinition
= image_attribute_redefinition
= interface_attribute_redefinition
= item_array_attribute_redefinition
= list_attribute_redefinition
= menu_attribute_redefinition
= method_attribute_redefinition
= program_attribute_redefinition
= record_attribute_redefinition
= source_attribute_redefinition
= template_attribute_redefinition
= variable_attribute_redefinition
= variable_list_attribute_redefinition
= waveform_attribute_redefinition

```

A.6.23 MENU

menu:

```
= 'MENU' Identifier '{' menu_attribute_list '}'
```

menu_attribute_list:

```
= menu_attribute
= menu_attribute_list menu_attribute
```

menu_attribute:

```

= menu_items           /* M */
= label                 /* M */
= menu_access           /* O */
= help                  /* O */
= post_edit_actions     /* O */
= post_read_actions     /* O */
= post_write_actions    /* O */
= pre_edit_actions      /* O */
= pre_read_actions      /* O */
= pre_write_actions     /* O */
= menu_purpose            /* O */
= menu_role              /* O */
= menu_style            /* O */
= validity              /* O */

```

menu_items:

```

= 'ITEMS' '{' '}'
= 'ITEMS' '{' menu_item_specifier_list '}'

```

menu_item_specifier_list:

```

= menu_item_specifier
= menu_item_specifier_list menu_item_specifier

```

menu_item_specifier:

```
= menu_item_list
```

```

= 'IF' '(' expr ')' '{' menu_item_specifier_list '}'
= 'IF' '(' expr ')' '{' menu_item_specifier_list '}'
  'ELSE' '{' menu_item_specifier_list '}'
= 'SELECT' '(' expr ')' '{' menu_item_selection_list '}'

menu_item_selection_list:
= menu_item_selection
= menu_item_selection_list menu_item_selection

menu_item_selection:
= 'CASE' expr ':' menu_item_specifier_list
= 'DEFAULT' ':' menu_item_specifier_list

menu_item_list:
= menu_item
= menu_item_list ',' menu_item

menu_item:
= reference_without_call
= reference_without_call menu_item_args
= 'SEPARATOR'
= 'COLUMNBREAK'
= 'ROWBREAK'

menu_item_args:
= '(' argument_list ')'
= '(' 'REVIEW' ')'
= '(' variable_qualifier_list ')'
= '(' image_qualifier_list ')'

variable_qualifier_list:
= variable_qualifier
= variable_qualifier_list ',' variable_qualifier

variable_qualifier:
= 'DISPLAY_VALUE'
= 'READ_ONLY'
= 'HIDDEN'
= 'NO_LABEL'
= 'NO_UNIT'

image_qualifier_list:
= image_qualifier
= image_qualifier_list ',' image_qualifier

image_qualifier:
= 'INLINE'

menu_access:
= 'ACCESS' 'ONLINE' ';'
= 'ACCESS' 'OFFLINE' ';'

menu_purpose:
= 'PURPOSE' menu_purpose_specifier

menu_purpose_specifier:
= menu_purpose_list ';'
= 'IF' '(' expr ')' '{' menu_purpose_specifier '}'
= 'IF' '(' expr ')' '{' menu_purpose_specifier '}'
  'ELSE' '{' menu_purpose_specifier '}'
= 'SELECT' '(' expr ')' '{' menu_purpose_selection_list '}'

menu_purpose_selection_list:

```

```

= menu_purpose_selection
= menu_purpose_selection_list menu_purpose_selection

menu_purpose_selection:
= 'CASE' expr ':' menu_purpose_specifier
= 'DEFAULT' ':' menu_purpose_specifier

menu_purpose_list:
= menu_purpose_item
= menu_purpose_list '&' menu_purpose_item

menu_purpose_item:
= 'CATALOG'
= 'DIAGNOSE'
= 'LOAD_TO_APPLICATION'
= 'LOAD_TO_DEVICE'
= 'MENU'
= 'PROCESS_VALUE'
= 'SIMULATION'
= 'TABLE'
= string

menu_role:
= 'ROLE' menu_role_specifier

menu_role_specifier:
= menu_role_list ';'
= 'IF' '(' expr ')' '{' menu_role_specifier '}'
= 'IF' '(' expr ')' '{' menu_role_specifier '}'
= 'ELSE' '{' menu_role_specifier '}'
= 'SELECT' '(' expr ')' '{' menu_role_selection_list '}'

menu_role_selection_list:
= menu_role_selection
= menu_role_selection_list menu_role_selection

menu_role_selection:
= 'CASE' expr ':' menu_role_specifier
= 'DEFAULT' ':' menu_role_specifier

menu_role_list:
= menu_role_item
= menu_role_list '&' menu_role_item

menu_role_item:
= string

menu_style:
= 'STYLE' menu_style_item ';'

menu_style_item:
= 'DIALOG'
= 'GROUP'
= 'MENU'
= 'PAGE'
= 'TABLE'
= 'WINDOW'
= string

```

A.6.24 METHOD

```

method:
= 'METHOD' Identifier '{' method_attribute_list '}'
= 'METHOD' Identifier method_parameters '{' method_attribute_list '}'

```

```

method_parameters:
    = '(' method_parameter_list ')'

method_parameter_list:
    = method_parameter
    = method_parameter_list ',' method_parameter

method_parameter:
    = method_parameter_type Identifier
    = method_parameter_type '&' Identifier
    = method_parameter_type Identifier '[' ']'

method_parameter_type:
    = 'char'
    = 'unsigned' 'char'
    = 'short'
    = 'unsigned' 'short'
    = 'int'
    = 'unsigned' 'int'
    = 'long'
    = 'unsigned' 'long'
    = 'float'
    = 'double'
    = 'time_t'
    = 'DD_STRING'
    = 'DD_ITEM'

method_attribute_list:
    = method_attribute
    = method_attribute_list method_attribute

method_attribute:
    = method_definition /* M */
    = method_access /* O */
    = variable_class /* O */
    = help /* O */
    = label /* O */
    = method_type /* O */
    = validity /* O */

method_definition:
    = 'DEFINITION' c_compound_statement

method_access:
    = 'ACCESS' 'OFFLINE' ';'
    = 'ACCESS' 'ONLINE' ';'

method_type:
    = 'TYPE' method_return_type

method_return_type:
    = 'char'
    = 'unsigned' 'char'
    = 'int'
    = 'unsigned' 'int'
    = 'long'
    = 'unsigned' 'long'
    = 'float'
    = 'double'
    = 'time_t'
    = 'DD_STRING'

```

A.6.25 PROGRAM

```

program:
  = 'PROGRAM' Identifier '{' program_attribute_list '}'
  = 'PROGRAM' Identifier '{' '}'

program_attribute_list:
  = program_attribute
  = program_attribute_list program_attribute

program_attribute:
  = arguments          /* O */
  = response_codes     /* O */

arguments:
  = 'ARGUMENTS' '{' '}'
  = 'ARGUMENTS' '{' arguments_specifier_list '}'

arguments_specifier_list:
  = arguments_specifier
  = arguments_specifier_list arguments_specifier

arguments_specifier:
  = argument_list
  = 'IF' '(' expr ')' '{' arguments_specifier_list '}'
  = 'IF' '(' expr ')' '{' arguments_specifier_list '}'
  = 'ELSE' '{' arguments_specifier_list '}'
  = 'SELECT' '(' expr ')' '{' arguments_selection_list '}'

arguments_selection_list:
  = arguments_selection
  = arguments_selection_list arguments_selection

arguments_selection:
  = 'CASE' expr ':' arguments_specifier_list
  = 'DEFAULT' ':' arguments_specifier_list

argument_list:
  = argument
  = argument_list ',' argument

argument:
  = Integer
  = variable_reference

```

A.6.26 RECORD

```

record:
  = 'RECORD' Identifier '{' record_attribute_list '}'

record_attribute_list:
  = record_attribute
  = record_attribute_list record_attribute

record_attribute:
  = label          /* M */
  = members        /* M */
  = help           /* O */
  = response_codes /* O */
  = validity       /* O */

```

A.6.27 REFERENCE_ARRAY

Reference_Array (matrice de référence) a deux mots clés disponibles, "ITEM_ARRAY" (première option) et "ARRAY" (deuxième option) qui peuvent être sélectionnés par un profil.

Le mot clé ARRAY ne doit pas être sélectionné si le paramètre value_array (matrice de valeurs) est utilisé.

NOTE Si une application EDD prend en charge plus d'un profil, le mot clé ARRAY peut également être utilisé à la place d'ITEM_ARRAY. Dans ce cas, la sélection sémantique est effectuée par des attributs.

```

item_array:
  = 'ITEM_ARRAY' 'OF' item_type Identifier
    '{' item_array_attribute_list '}'
  = 'ARRAY' 'OF' item_type Identifier
    '{' item_array_attribute_list '}'

item_array_attribute_list:
  = item_array_attribute
  = item_array_attribute_list item_array_attribute

item_array_attribute:
  = elements          /* M */
  = help              /* O */
  = label             /* O */
  = validity          /* O */

elements:
  = 'ELEMENTS' '{' elements_specifier_list '}'

elements_specifier_list:
  = elements_specifier
  = elements_specifier_list elements_specifier

elements_specifier:
  = element
  = 'IF' '(' expr ')' '{' elements_specifier_list '}'
  = 'IF' '(' expr ')' '{' elements_specifier_list '}'
  = 'ELSE' '{' elements_specifier_list '}'
  = 'SELECT' '(' expr ')' '{' elements_selection_list '}'

elements_selection_list:
  = elements_selection
  = elements_selection_list elements_selection

elements_selection:
  = 'CASE' expr ':' elements_specifier_list
  = 'DEFAULT' ':' elements_specifier_list

element:
  = Integer ',' reference_without_call ';'
  = Integer ',' reference_without_call ',' description_string ';'
  = Integer ',' reference_without_call ',' description_string
    ',' help_string ';'

```

A.6.28 Relations

```

refresh_relation:
  = 'REFRESH' Identifier
    '{' refresh_relation_left ':' refresh_relation_right '}'

refresh_relation_left:
  = variable_reference_specifier_list

refresh_relation_right:
  = variable_reference_specifier_list

unit_relation:

```



```

= 'UNIT' Identifier
  '{' unit_relation_left ':' unit_relation_right '}'

unit_relation_left:
= variable_reference_specifier

unit_relation_right:
= variable_axis_reference_selection_list

wao_relation:
= 'WRITE_AS_ONE' Identifier '{' wao_specifier '}'

wao_specifier:
= variable_reference_specifier_list

variable_reference_specifier_list:
= variable_reference_specifier
= variable_reference_specifier_list variable_reference_specifier
= variable_reference_specifier_list ',' variable_reference_specifier

variable_reference_specifier:
= variable_reference
= 'IF' '(' expr ')' '{' variable_reference_specifier '}'
= 'IF' '(' expr ')' '{' variable_reference_specifier '}'
  'ELSE' '{' variable_reference_specifier '}'
= 'SELECT' '(' expr ')' '{' variable_reference_selection_list '}'

variable_reference_selection_list:
= variable_reference_selection
= variable_reference_selection_list variable_reference_selection

variable_reference_selection:
= 'CASE' expr ':' variable_reference_specifier
= 'DEFAULT' ':' variable_reference_specifier

variable_axis_reference_selection_list:
= variable_axis_reference_selection
= variable_axis_reference_selection_list variable_axis_reference_selection

variable_axis_reference_selection:
= 'CASE' expr ':' variable_axis_reference_specifier
= 'DEFAULT' ':' variable_axis_reference_specifier

variable_axis_reference_specifier:
= variable_reference
= axis_reference
= 'IF' '(' expr ')' '{' variable_axis_reference_specifier '}'
= 'IF' '(' expr ')' '{' variable_axis_reference_specifier '}'
  'ELSE' '{' variable_axis_reference_specifier '}'
= 'SELECT' '(' expr ')' '{' variable_axis_reference_selection_list '}'

```

A.6.29 RESPONSE_CODES

```

response_codes_definition:
= 'RESPONSE_CODES' Identifier '{' response_codes_attribute_list
  '}'

response_codes_attribute_list:
= response_codes_specifier_list

response_codes_specifier_list:
= response_codes_specifier
= response_codes_specifier_list response_codes_specifier

```

```
response_codes_specifier:
= response_code
= 'IF' '(' expr ')' '{' response_codes_specifier_list '}'
= 'IF' '(' expr ')' '{' response_codes_specifier_list '}'
  'ELSE' '{' response_codes_specifier_list '}'
= 'SELECT' '(' expr ')' '{' response_codes_selection_list '}'
```

```
response_codes_selection_list:
= response_codes_selection
= response_codes_selection_list response_codes_selection
```

```
response_codes_selection:
= 'CASE' expr ':' response_codes_specifier_list
= 'DEFAULT' ':' response_codes_specifier_list
```

```
response_code:
= Integer ',' response_code_type ',' description_string ';'
= Integer ',' response_code_type ',' description_string ','
  help_string ';'

```

```
response_code_type:
= 'SUCCESS'
= 'MISC_WARNING'
= 'DATA_ENTRY_WARNING'
= 'DATA_ENTRY_ERROR'
= 'MODE_ERROR'
= 'PROCESS_ERROR'
= 'MISC_ERROR'
```

A.6.30 SOURCE

```
source:
= 'SOURCE' Identifier '{' source_attribute_list '}'
```

```
source_attribute_list:
= source_attribute
= source_attribute_list source_attribute
```

```
source_attribute:
= members /* M */
= emphasis /* O */
= source_exit_actions /* O */
= help /* O */
= source_init_actions /* O */
= label /* O */
= line_color /* O */
= line_type /* O */
= source_refresh_actions /* O */
= validity /* O */
= source_y_axis /* O */

```

```
source_exit_actions:
= 'EXIT_ACTIONS' '{' actions_specifier_list '}'
```

```
source_init_actions:
= 'INIT_ACTIONS' '{' actions_specifier_list '}'
```

```
source_refresh_actions:
= 'REFRESH_ACTIONS' '{' actions_specifier_list '}'
```

```
source_y_axis:
= 'Y_AXIS' axis_specifier
```

A.6.31 TEMPLATE

```

template:
  = 'TEMPLATE' Identifier '{' template_attribute_list '}'

template_attribute_list:
  = template_attribute
  = template_attribute_list template_attribute

template_attribute:
  = template_default_values      /* M */
  = label                       /* M */
  = help                       /* O */

template_default_values:
  = 'DEFAULT_VALUES' '{' template_default_value_list '}'

template_default_value_list:
  = template_default_value
  = template_default_value_list template_default_value

template_default_value:
  = reference_without_call '=' expr_specifier

```

A.6.32 VALUE_ARRAY

```

array:
  = 'ARRAY' Identifier '{' array_attribute_list '}'

array_attribute_list:
  = array_attribute
  = array_attribute_list array_attribute

array_attribute:
  = label                /* M */
  = array_size           /* M */
  = array_type           /* M */
  = help                /* O */
  = response_codes       /* O */
  = validity             /* O */

array_size:
  = 'NUMBER_OF_ELEMENTS' Integer ';'

array_type:
  = 'TYPE' Identifier ';'

```

A.6.33 VARIABLE

```

variable:
  = 'VARIABLE' Identifier '{' variable_attribute_list '}'

variable_attribute_list:
  = variable_attribute
  = variable_attribute_list variable_attribute

variable_attribute:
  = variable_class      /* M */
  = type                /* M */
  = constant_unit       /* O */
  = default_value       /* O */
  = handling            /* O */
  = help                /* O */
  = initial_value       /* O */
  = label               /* O */

```

```
= post_edit_actions      /* 0 */
= post_read_actions      /* 0 */
= post_write_actions     /* 0 */
= pre_edit_actions       /* 0 */
= pre_read_actions       /* 0 */
= pre_write_actions      /* 0 */
= read_timeout           /* 0 */
= refresh_actions        /* 0 */
= response_codes         /* 0 */
= variable_style         /* 0 */
= validity               /* 0 */
= write_timeout          /* 0 */
```

```
variable_class:
= 'CLASS' variable_class_definition ';'

```

```
variable_class_definition:
= variable_class_keyword
= variable_class_definition '&' variable_class_keyword

```

```
variable_class_keyword:
```

```
= 'ALARM'
= 'ANALOG_INPUT'
= 'ANALOG_OUTPUT'
= 'COMPUTATION'
= 'CONTAINED'
= 'CORRECTION'
= 'DEVICE'
= 'DIAGNOSTIC'
= 'DIGITAL_INPUT'
= 'DIGITAL_OUTPUT'
= 'DISCRETE_INPUT'
= 'DISCRETE_OUTPUT'
= 'DYNAMIC'
= 'FREQUENCY_INPUT'
= 'FREQUENCY_OUTPUT'
= 'HART'
= 'INPUT'
= 'LOCAL'
= 'LOCAL_DISPLAY'
= 'OPERATE'
= 'OPTIONAL'
= 'OUTPUT'
= 'SERVICE'
= 'TEMPORARY'
= 'TUNE'
```

```
type:
```

```
= 'TYPE' type_specifier

```

```
type_specifier:
```

```
= arithmetic_type
= enumerated_type
= index_type
= string_type
= date_time_type
= object_reference

```

```
arithmetic_type:
```

```
= 'INTEGER' arithmetic_size arithmetic_options
= 'UNSIGNED_INTEGER' arithmetic_size arithmetic_options
= 'DOUBLE' arithmetic_options
= 'FLOAT' arithmetic_options

```

```

arithmetic_size:
=
= '(' Integer ')'

arithmetic_options:
= ';'
= '{' arithmetic_option_list '}'

arithmetic_option_list:
= arithmetic_option
= arithmetic_option_list arithmetic_option

arithmetic_option:
= default_value
= initial_value
= display_format
= edit_format
= arithmetic_enumerator_list
= maximum_value
= maximum_value_n
= minimum_value
= minimum_value_n
= scaling_factor

display_format:
= 'DISPLAY_FORMAT' string_specifier

edit_format:
= 'EDIT_FORMAT' string_specifier

arithmetic_enumerator_list:
= arithmetic_enumerator
= arithmetic_enumerator_list ',' arithmetic_enumerator

arithmetic_enumerator:
= '{' arithmetic_value ',' description_string '}'
= '{' arithmetic_value ',' description_string ',' help_string '}'

arithmetic_value:
= Integer
= Real

scaling_factor:
= 'SCALING_FACTOR' expr_specifier

maximum_value:
= 'MAX_VALUE' expr_specifier

maximum_value_n:
= 'MAX_VALUE1' expr_specifier
= 'MAX_VALUE2' expr_specifier
= 'MAX_VALUE3' expr_specifier
= 'MAX_VALUE4' expr_specifier
= 'MAX_VALUE5' expr_specifier
= 'MAX_VALUE6' expr_specifier
= 'MAX_VALUE7' expr_specifier
= 'MAX_VALUE8' expr_specifier
= 'MAX_VALUE9' expr_specifier

minimum_value:
= 'MIN_VALUE' expr_specifier

```

```

minimum_value_n:
= 'MIN_VALUE1' expr_specifier
= 'MIN_VALUE2' expr_specifier
= 'MIN_VALUE3' expr_specifier
= 'MIN_VALUE4' expr_specifier
= 'MIN_VALUE5' expr_specifier
= 'MIN_VALUE6' expr_specifier
= 'MIN_VALUE7' expr_specifier
= 'MIN_VALUE8' expr_specifier
= 'MIN_VALUE9' expr_specifier

enumerated_type:
= 'ENUMERATED' enumerated_size '{' enumerated_option_list
  '}'
= 'BIT_ENUMERATED' enumerated_size '{'
  bit_enumerated_option_list '}'

enumerated_size:
=
= '(' Integer ')'

enumerated_option_list:
= enumerated_option
= enumerated_option_list enumerated_option

enumerated_option:
= default_value
= initial_value
= enumerators_specifier

enumerators_specifier_list:
= enumerators_specifier
= enumerators_specifier_list enumerators_specifier

enumerators_specifier:
= integral_enumerator_list
= 'IF' '(' expr ')' '{' enumerators_specifier_list '}'
= 'IF' '(' expr ')' '{' enumerators_specifier_list '}'
  'ELSE' '{' enumerators_specifier_list '}'
= 'SELECT' '(' expr ')' '{' enumerators_selection_list '}'

enumerators_selection_list:
= enumerators_selection
= enumerators_selection_list enumerators_selection

enumerators_selection:
= 'CASE' expr ':' enumerators_specifier_list
= 'DEFAULT' ':' enumerators_specifier_list

integral_enumerator_list:
= integral_enumerator
= integral_enumerator_list ',' integral_enumerator

integral_enumerator:
= '{' Integer ',' description_string '}'
= '{' Integer ',' description_string ',' help_string '}'

description_string:
= string

help_string:
= string

```

```

bit_enumerated_option_list:
  = bit_enumerated_option
  = bit_enumerated_option_list bit_enumerated_option

bit_enumerated_option:
  = default_value
  = initial_value
  = bit_enumerators_specifier

bit_enumerators_specifier_list:
  = bit_enumerators_specifier
  = bit_enumerators_specifier_list bit_enumerators_specifier

bit_enumerators_specifier:
  = bit_enumerator_list
  = 'IF' '(' expr ')' '{' bit_enumerators_specifier_list '}'
  = 'IF' '(' expr ')' '{' bit_enumerators_specifier_list '}'
    'ELSE' '{' bit_enumerators_specifier_list '}'
  = 'SELECT' '(' expr ')' '{' bit_enumerators_selection_list '}'

bit_enumerators_selection_list:
  = bit_enumerators_selection
  = bit_enumerators_selection_list bit_enumerators_selection

bit_enumerators_selection:
  = 'CASE' expr ':' bit_enumerators_specifier_list
  = 'DEFAULT' ':' bit_enumerators_specifier_list

bit_enumerator_list:
  = bit_enumerator
  = bit_enumerator_list ',' bit_enumerator

bit_enumerator:
  = '{' Integer ',' string '}'
  = '{' Integer ',' string ',' help_string '}'
  = '{' Integer ',' string ',' variable_class_definition '}'
  = '{' Integer ',' string ',' status_class '}'
  = '{' Integer ',' string ',' string ',' variable_class_definition '}'
  = '{' Integer ',' string ',' string ',' status_class '}'
  = '{' Integer ',' string ',' string ',' method_reference '}'
  = '{' Integer ',' string ',' variable_class_definition ','
    status_class '}'
  = '{' Integer ',' string ',' variable_class_definition ','
    method_reference '}'
  = '{' Integer ',' string ',' status_class ',' method_reference '}'
  = '{' Integer ',' string ',' string ',' variable_class_definition ','
    status_class '}'
  = '{' Integer ',' string ',' string ',' variable_class_definition ','
    method_reference '}'
  = '{' Integer ',' string ',' string ',' status_class ','
    method_reference '}'
  = '{' Integer ',' string ',' variable_class_definition ',' status_class
    ',' method_reference '}'
  = '{' Integer ',' string ',' string ',' variable_class_definition ','
    status_class ',' method_reference '}'

status_class:
  = status_class_definition

status_class_definition:
  = status_class_keyword
  = status_class_definition '&' status_class_keyword

```

```

status_class_keyword:
= 'HARDWARE'
= 'SOFTWARE'
= 'PROCESS'
= 'MODE'
= 'DATA'
= 'MISC'
= 'EVENT'
= 'STATE'
= 'SELF_CORRECTING'
= 'CORRECTABLE'
= 'UNCORRECTABLE'
= 'SUMMARY'
= 'DETAIL'
= 'MORE'
= 'COMM'
= 'IGNORE_IN_TEMPORARY_MASTER'
= 'BAD'
= 'INFO'
= 'WARNING'
= 'IGNORE_IN_HOST'

index_type:
= 'INDEX' item_array_reference ';'
= 'INDEX' '(' Integer ')' item_array_reference ';'

string_type:
= 'ASCII' string_size string_options
= 'BITSTRING' string_size string_options
= 'EUC' string_size string_options
= 'PACKED_ASCII' string_size string_options
= 'PASSWORD' string_size string_options
= 'VISIBLE' string_size string_options
= 'OCTET' string_size string_options

string_size:
= '(' Integer ')'

string_options:
= ';'
= '{' string_option_list '}'

string_option_list:
= string_option
= string_option_list string_option

string_option:
= string_default_value
= string_initial_value
= string_enumerator_list

string_enumerator_list:
= string_enumerator
= string_enumerator_list ',' string_enumerator

string_enumerator:
= '{' string ',' description_string '}'
= '{' string ',' description_string ',' help_string '}'

string_default_value:
= 'DEFAULT_VALUE' string_specifier

string_initial_value:

```



```

= 'INITIAL_VALUE' string ';'

date_time_type:
= 'DATE' date_time_options
= 'DATE_AND_TIME' date_time_options
= 'DURATION' date_time_options
= 'TIME' date_time_options
= 'TIME_VALUE' date_time_size time_value_options

date_time_size:
=
= '(' Integer ')'

date_time_options:
= ';'
= '{' date_time_option_list '}'

date_time_option_list:
= date_time_option
= date_time_option_list date_time_option

date_time_option:
= default_value
= initial_value

time_value_options:
= ';'
= '{' time_value_option_list '}'

time_value_option_list:
= time_value_option
= time_value_option_list time_value_option

time_value_option:
= default_value
= display_format
= edit_format
= initial_value
= time_format
= time_scale

time_format:
= 'TIME_FORMAT' string_specifier

time_scale:
= 'TIME_SCALE' time_scale_specifier

time_scale_specifier:
= time_scale_keyword ';'
= 'IF' '(' expr ')' '{' time_scale_specifier '}'
= 'IF' '(' expr ')' '{' time_scale_specifier '}'
  'ELSE' '{' time_scale_specifier '}'
= 'SELECT' '(' expr ')' '{' time_scale_selection_list '}'

time_scale_selection_list:
= time_scale_selection
= time_scale_selection_list time_scale_selection

time_scale_selection:
= 'CASE' expr ':' time_scale_specifier
= 'DEFAULT' ':' time_scale_specifier

time_scale_keyword:

```

```

= 'SECONDS'
= 'MINUTES'
= 'HOURS'

object_reference:
= 'OBJECT_REFERENCE' object_reference_value ';'

object_reference_value:
= 'ROOT'
= 'PARENT'
= 'CHILD'
= 'SELF'
= 'NEXT'
= 'PREV'
= 'FIRST'
= 'LAST'

constant_unit:
= 'CONSTANT_UNIT' string_specifier

default_value:
= 'DEFAULT_VALUE' expr_specifier

initial_value:
= 'INITIAL_VALUE' expr ';'

pre_edit_actions:
= 'PRE_EDIT_ACTIONS' '{' actions_specifier_list '}'

post_edit_actions:
= 'POST_EDIT_ACTIONS' '{' actions_specifier_list '}'

pre_read_actions:
= 'PRE_READ_ACTIONS' '{' actions_specifier_list '}'

post_read_actions:
= 'POST_READ_ACTIONS' '{' actions_specifier_list '}'

pre_write_actions:
= 'PRE_WRITE_ACTIONS' '{' actions_specifier_list '}'

post_write_actions:
= 'POST_WRITE_ACTIONS' '{' actions_specifier_list '}'

read_timeout:
= 'READ_TIMEOUT' expr_specifier

refresh_actions:
= 'REFRESH_ACTIONS' '{' actions_specifier_list '}'

actions_specifier_list:
= method_specification
= actions_specifier_list ',' method_specification

method_specification:
= method_reference
= method_definition

variable_style:
= 'STYLE' string ';'

write_timeout:
= 'WRITE_TIMEOUT' expr_specifier

```

```

expr_specifier:
= expr ';'
= 'IF' '(' expr ')' '{' expr_specifier '}'
= 'IF' '(' expr ')' '{' expr_specifier '}'
  'ELSE' '{' expr_specifier '}'
= 'SELECT' '(' expr ')' '{' expr_selection_list '}'

```

```

expr_selection_list:
= expr_selection
= expr_selection_list expr_selection

```

```

expr_selection:
= 'CASE' expr ':' expr_specifier
= 'DEFAULT' ':' expr_specifier

```

A.6.34 VARIABLE_LIST

```

variable_list:
= 'VARIABLE_LIST' Identifier '{' variable_list_attribute_list '}'

```

```

variable_list_attribute_list:
= variable_list_attribute
= variable_list_attribute_list variable_list_attribute

```

```

variable_list_attribute:
= members          /* M */
= help             /* O */
= label            /* O */
= response_codes   /* O */

```

A.6.35 WAVEFORM

```

waveform:
= 'WAVEFORM' Identifier '{' waveform_attribute_list '}'

```

```

waveform_attribute_list:
= waveform_attribute
= waveform_attribute_list waveform_attribute

```

```

waveform_attribute:
= waveform_type          /* M */
= emphasis              /* O */
= waveform_exit_actions /* O */
= handling              /* O */
= help                  /* O */
= waveform_init_actions /* O */
= waveform_key_points   /* O */
= label                 /* O */
= line_color            /* O */
= line_type             /* O */
= waveform_refresh_actions /* O */
= validity              /* O */
= waveform_y_axis       /* O */

```

```

waveform_type:
= 'TYPE' waveform_type_specifier

```

```

waveform_type_specifier:
= waveform_yt_type
= waveform_xy_type
= waveform_horizontal_type
= waveform_vertical_type

```

```

waveform_yt_type:
    = 'YT' '{' waveform_yt_type_option_list '}'

waveform_yt_type_option_list:
    = waveform_yt_type_option
    = waveform_yt_type_option_list waveform_yt_type_option

waveform_yt_type_option:
    = yt_x_initial          /* M */
    = yt_x_increment        /* M */
    = waveform_y_values      /* M */
    = number_of_points      /* O */

number_of_points:
    = 'NUMBER_OF_POINTS' expr_specifier

yt_x_initial:
    = 'X_INITIAL' expr_specifier

yt_x_increment:
    = 'X_INCREMENT' expr_specifier

waveform_y_values:
    = 'Y_VALUES' '{' valeurs_option_list '}'

waveform_x_values:
    = 'X_VALUES' '{' valeurs_option_list '}'

valeurs_option_list:
    = valeurs_option
    = valeurs_option_list ',' valeurs_option

valeurs_option:
    = reference_without_call
    = Integer
    = Real

waveform_xy_type:
    = 'XY' '{' waveform_xy_type_option_list '}'

waveform_xy_type_option_list:
    = waveform_xy_type_option
    = waveform_xy_type_option_list waveform_xy_type_option

waveform_xy_type_option:
    = waveform_x_values      /* M */
    = waveform_y_values      /* M */
    = number_of_points      /* O */

waveform_horizontal_type:
    = 'HORIZONTAL' '{' waveform_y_values '}'

waveform_vertical_type:
    = 'VERTICAL' '{' waveform_x_values '}'

waveform_y_axis:
    = 'Y_AXIS' axis_specifier

axis_specifier:
    = axis_reference ';'
    = 'IF' '(' expr ')' '{' axis_specifier '}'
    = 'IF' '(' expr ')' '{' axis_specifier '}'
    'ELSE' '{' axis_specifier '}'

```

```

= 'SELECT' '(' expr ')' '{' axis_selection_list '}'

axis_selection_list:
= axis_selection
= axis_selection_list axis_selection

axis_selection:
= 'CASE' expr ':' axis_specifier
= 'DEFAULT' ':' axis_specifier

waveform_exit_actions:
= 'EXIT_ACTIONS' '{' actions_specifier_list '}'

waveform_init_actions:
= 'INIT_ACTIONS' '{' actions_specifier_list '}'

waveform_key_points:
= 'KEY_POINTS' '{' key_points_options_list '}'

key_points_options_list:
= key_points_options
= key_points_options_list key_points_options

key_points_options:
= waveform_x_values      /* M */
= waveform_y_values      /* M */

waveform_refresh_actions:
= 'REFRESH_ACTIONS' '{' actions_specifier_list '}'

```

A.6.36 Attributs communs

```

emphasis:
= 'EMPHASIS' boolean_specifier

boolean_specifier:
= boolean ';'
= 'IF' '(' expr ')' '{' boolean_specifier '}'
= 'IF' '(' expr ')' '{' boolean_specifier '}'
  'ELSE' '{' boolean_specifier '}'
= 'SELECT' '(' expr ')' '{' boolean_selection_list '}'

boolean_selection_list:
= boolean_selection
= boolean_selection_list boolean_selection

boolean_selection:
= 'CASE' expr ':' boolean_specifier
= 'DEFAULT' ':' boolean_specifier

boolean:
= 'FALSE'
= 'TRUE'

handling:
= 'HANDLING' handling_specifier

handling_specifier:
= handling_definition ';'
= 'IF' '(' expr ')' '{' handling_specifier '}'
= 'IF' '(' expr ')' '{' handling_specifier '}'
  'ELSE' '{' handling_specifier '}'
= 'SELECT' '(' expr ')' '{' handling_selection_list '}'

```

```

handling_selection_list:
    = handling_selection
    = handling_selection_list handling_selection

handling_selection:
    = 'CASE' expr ':' handling_specifier
    = 'DEFAULT' ':' handling_specifier

handling_definition:
    = handling_keyword
    = handling_definition '&' handling_keyword

handling_keyword:
    = 'READ'
    = 'WRITE'

height:
    = 'HEIGHT' display_size_keyword ';'

display_size_keyword:
    = 'XX_SMALL'
    = 'X_SMALL'
    = 'SMALL'
    = 'MEDIUM'
    = 'XX_LARGE'
    = 'X_LARGE'
    = 'LARGE'

help:
    = 'HELP' string_specifier

string_specifier:
    = string ';'
    = 'IF' '(' expr ')' '{' string_specifier '}'
    = 'IF' '(' expr ')' '{' string_specifier '}'
    = 'ELSE' '{' string_specifier '}'
    = 'SELECT' '(' expr ')' '{' string_selection_list '}'

string_selection_list:
    = string_selection
    = string_selection_list string_selection

string_selection:
    = 'CASE' expr ':' string_specifier
    = 'DEFAULT' ':' string_specifier

string:
    = string_list

string_list:
    = string_terminal
    = string_list '+' string_terminal

string_terminal:
    = string_literal
    = variable_reference
    = variable_reference '(' Integer ')'
    = dictionary_reference

string_literal:
    = String
    = string_literal String

```

```

label:
    = 'LABEL' string_specifier

line_color:
    = 'LINE_COLOR' line_color_specifier

line_color_specifier:
    = color ';'
    = 'IF' '(' expr ')' '{' line_color_specifier '}'
    = 'IF' '(' expr ')' '{' line_color_specifier '}'
    = 'ELSE' '{' line_color_specifier '}'
    = 'SELECT' '(' expr ')' '{' line_color_selection_list '}'

line_color_selection_list:
    = line_color_selection
    = line_color_selection_list line_color_selection

line_color_selection:
    = 'CASE' expr ':' line_color_specifier
    = 'DEFAULT' ':' line_color_specifier

color:
    = Integer
    = variable_reference

line_type:
    = 'LINE_TYPE' line_type_specifier

line_type_specifier:
    = line_type_keyword ';'
    = 'IF' '(' expr ')' '{' line_type_specifier '}'
    = 'IF' '(' expr ')' '{' line_type_specifier '}'
    = 'ELSE' '{' line_type_specifier '}'
    = 'SELECT' '(' expr ')' '{' line_type_selection_list '}'

line_type_selection_list:
    = line_type_selection
    = line_type_selection_list line_type_selection

line_type_selection:
    = 'CASE' expr ':' line_type_specifier
    = 'DEFAULT' ':' line_type_specifier

line_type_keyword:
    = 'DATA'
    = 'DATA1'
    = 'DATA2'
    = 'DATA3'
    = 'DATA4'
    = 'DATA5'
    = 'DATA6'
    = 'DATA7'
    = 'DATA8'
    = 'DATA9'
    = 'LOW_LOW_LIMIT'
    = 'LOW_LIMIT'
    = 'HIGH_LIMIT'
    = 'HIGH_HIGH_LIMIT'
    = 'TRANSPARENT'

members:
    = 'MEMBERS' '{' members_specifier_list '}'

```

```

members_specifier_list:
  = members_specifier
  = members_specifier_list members_specifier

members_specifier:
  = member
  = 'IF' '(' expr ')' '{' members_specifier_list '}'
  = 'IF' '(' expr ')' '{' members_specifier_list '}'
  = 'ELSE' '{' members_specifier_list '}'
  = 'SELECT' '(' expr ')' '{' members_selection_list '}'

members_selection_list:
  = members_selection
  = members_selection_list members_selection

members_selection:
  = 'CASE' expr ':' members_specifier_list
  = 'DEFAULT' ':' members_specifier_list

member:
  = Identifier ',' reference_without_call ';'
  = Identifier ',' reference_without_call '(' string ','
  = Identifier ',' reference_without_call '(' string ',' string ';'

response_codes:
  = 'RESPONSE_CODES' '{' response_codes_specifier_list '}'
  = 'RESPONSE_CODES' response_code_reference_specifier

response_code_reference_specifier:
  = response_code_reference
  = 'IF' '(' expr ')' '{' response_code_reference_specifier '}'
  = 'IF' '(' expr ')' '{' response_code_reference_specifier
    '}'
  = 'ELSE' '{' response_code_reference_specifier '}'
  = 'SELECT' '(' expr ')' '{'
    response_code_reference_selection_list '}'

response_code_reference_selection_list:
  = response_code_reference_selection
  = response_code_reference_selection_list
    response_code_reference_selection

response_code_reference_selection:
  = 'CASE' expr ':' response_code_reference_specifier
  = 'DEFAULT' ':' response_code_reference_specifier

validity:
  = 'VALIDITY' boolean_specifier

largeur:
  = 'WIDTH' display_size_keyword ';'

```

A.6.37 Expression

```

primary_expr:
  = Integer
  = Real
  = String
  = variable_reference
  = 'ARRAY_INDEX'
  = 'CURRENT_ROLE'
  = 'PI'

```



```
= '(' expr ')'
```

```
postfix_expr:
= primary_expr
= Identifier '(' ')'
= Identifier '(' c_argument_expr_list ')'
= axis_reference '.' axis_attribute_selector
= list_reference '.' list_attribute_selector
= variable_reference '.' variable_attribute_selector
```

```
axis_attribute_selector:
= 'MAX_VALUE'
= 'MIN_VALUE'
= 'SCALING'
= 'VIEW_MAX'
= 'VIEW_MIN'
```

```
list_attribute_selector:
= 'CAPACITY'
= 'COUNT'
```

```
variable_attribute_selector:
= 'DEFAULT_VALUE'
= 'INITIAL_VALUE'
= 'MAX_VALUE'
= 'MIN_VALUE'
= 'SCALING_FACTOR'
= 'VARIABLE_STATUS'
```

```
unary_expr:
= postfix_expr
= unary_operator unary_expr
```

```
unary_operator:
= '+'
= '-'
= '~'
= '!'
```

```
multiplicative_expr:
= unary_expr
= multiplicative_expr '*' unary_expr
= multiplicative_expr '/' unary_expr
= multiplicative_expr '%' unary_expr
```

```
additive_expr:
= multiplicative_expr
= additive_expr '+' multiplicative_expr
= additive_expr '-' multiplicative_expr
```

```
shift_expr:
= additive_expr
= shift_expr '<<' additive_expr
= shift_expr '>>' additive_expr
```

```
relational_expr:
= shift_expr
= relational_expr '<' shift_expr
= relational_expr '>' shift_expr
= relational_expr '<=' shift_expr
= relational_expr '>=' shift_expr
```

```
equality_expr:
```

```

= relational_expr
= equality_expr '==' relational_expr
= equality_expr '!=' relational_expr

and_expr:
= equality_expr
= and_expr '&' equality_expr

exclusive_or_expr:
= and_expr
= exclusive_or_expr '^' and_expr

inclusive_or_expr:
= exclusive_or_expr
= inclusive_or_expr '|' exclusive_or_expr

logical_and_expr:
= inclusive_or_expr
= logical_and_expr '&&' inclusive_or_expr

logical_or_expr:
= logical_and_expr
= logical_or_expr '||' logical_and_expr

conditional_expr:
= logical_or_expr
= logical_or_expr '?' expr ':' conditional_expr

expr:
= conditional_expr

```

A.6.38 Syntaxe C

```

c_primary_expr:
= c_char
= c_dictionary
= c_identifier
= c_integer
= c_real
= c_string
= c_reference
= '(' c_expr ')'

c_char:
= Character

c_dictionary:
= '[' Identifier ']'

c_identifier:
= Identifier

c_integer:
= Integer

c_real:
= Real

c_string:
= c_string_literal

c_string_literal:
= string_literal

```

```
c_reference:
= reference

c_postfix_expr:
= c_primary_expr
= c_postfix_expr '[' c_expr ']'
= Identifier '(' ')'
= Identifier '(' c_argument_expr_list ')'
= c_postfix_expr '.' Identifier
= c_postfix_expr '.' selector
= c_postfix_expr '++'
= c_postfix_expr '--'

c_argument_expr_list:
= c_assignment_expr
= c_argument_expr_list ',' c_assignment_expr

c_unary_expr:
= c_postfix_expr
= '++' c_unary_expr
= '--' c_unary_expr
= c_unary_operator c_postfix_expr

c_unary_operator:
= '+'
= '-'
= '~'
= '!'

c_multiplicative_expr:
= c_unary_expr
= c_multiplicative_expr '*' c_unary_expr
= c_multiplicative_expr '/' c_unary_expr
= c_multiplicative_expr '%' c_unary_expr

c_additive_expr:
= c_multiplicative_expr
= c_additive_expr '+' c_multiplicative_expr
= c_additive_expr '-' c_multiplicative_expr

c_shift_expr:
= c_additive_expr
= c_shift_expr '<<' c_additive_expr
= c_shift_expr '>>' c_additive_expr

c_relational_expr:
= c_shift_expr
= c_relational_expr '<' c_shift_expr
= c_relational_expr '>' c_shift_expr
= c_relational_expr '<=' c_shift_expr
= c_relational_expr '>=' c_shift_expr

c_equality_expr:
= c_relational_expr
= c_equality_expr '==' c_relational_expr
= c_equality_expr '!=' c_relational_expr

c_and_expr:
= c_equality_expr
= c_and_expr '&' c_equality_expr

c_exclusive_or_expr:
= c_and_expr
```

```

= c_exclusive_or_expr '^' c_and_expr

c_inclusive_or_expr:
= c_exclusive_or_expr
= c_inclusive_or_expr '|' c_exclusive_or_expr

c_logical_and_expr:
= c_inclusive_or_expr
= c_logical_and_expr '&&' c_inclusive_or_expr

c_logical_or_expr:
= c_logical_and_expr
= c_logical_or_expr '||' c_logical_and_expr

c_conditional_expr:
= c_logical_or_expr
= c_logical_or_expr '?' c_logical_or_expr ':' c_conditional_expr

c_assignment_expr:
= c_conditional_expr
= c_unary_expr c_assignment_operator c_assignment_expr

c_assignment_operator:
= '='
= '*='
= '/='
= '%='
= '+='
= '-='
= '>>='
= '<<='
= '&='
= '^='
= '|='

c_expr:
= c_assignment_expr
= c_expr ',' c_assignment_expr

c_constant_expr:
= c_conditional_expr

c_declaration:
= c_declaration_specifiers c_declarator_list ';'

c_declaration_specifiers:
= c_type_specifier
= c_declaration_specifiers c_type_specifier

c_type_specifier:
= 'char'
= 'short'
= 'int'
= 'long'
= 'signed'
= 'unsigned'
= 'float'
= 'double'
= 'time_t'
= 'DD_STRING'

c_declarator_list:
= c_declarator

```

```

= c_declarator_list ',' c_declarator

c_declarator:
= Identifier
= Identifier c_declarator_array_specifier_list

c_declarator_array_specifier_list:
= c_declarator_array_specifier
= c_declarator_array_specifier_list c_declarator_array_specifier

c_declarator_array_specifier:
= '[' ']'
= '[' c_constant_expr ']'

c_statement:
= c_labeled_statement
= c_compound_statement
= c_expr_statement
= c_selection_statement
= c_iteration_statement
= c_jump_statement

c_labeled_statement:
= 'case' c_constant_expr ':' c_statement
= 'default' ':' c_statement

c_compound_statement:
= '{' '}'
= '{' c_statement_list '}'
= '{' c_declaration_list '}'
= '{' c_declaration_list c_statement_list '}'

c_declaration_list:
= c_declaration
= c_declaration_list c_declaration

c_statement_list:
= c_statement
= c_statement_list c_statement

c_expr_statement:
= ';'
= c_expr ';'

c_selection_statement:
= 'if' '(' c_expr ')' c_statement
= 'if' '(' c_expr ')' c_statement 'else' c_statement
= 'switch' '(' c_expr ')' c_statement

c_iteration_statement:
= 'do' c_statement 'while' '(' c_expr ')' ';'
= 'while' '(' c_expr ')' c_statement
= 'for' '(' ';' ';' ';' ')'
  c_statement
= 'for' '(' ';' ';' c_expr ')'
  c_statement
= 'for' '(' ';' c_expr ';' ')'
  c_statement
= 'for' '(' ';' c_expr ';' c_expr ')'
  c_statement
= 'for' '(' c_expr ';' ';' ')'
  c_statement
= 'for' '(' c_expr ';' ';' c_expr ')'

```

```

c_statement
= 'for' '(' c_expr ';' c_expr ';' ')'
c_statement
= 'for' '(' c_expr ';' c_expr ';' c_expr ')'
c_statement

```

```

c_jump_statement:
= 'continue' ';'
= 'break' ';'
= 'return' ';'
= 'return' c_expr ';'

```

A.6.39 Redéfinition

```

array_redefinition:
= 'DELETE' 'ARRAY' Identifier ';'
= 'REDEFINE' 'ARRAY' Identifier '{' array_attribute_list '}'
= 'ARRAY' Identifier '{' array_attribute_redefinition_list '}'

array_attribute_redefinition_list:
= array_attribute_redefinition
= array_attribute_redefinition_list array_attribute_redefinition

array_attribute_redefinition:
= array_size_redefinition
= array_type_redefinition
= help_redefinition
= required_label_redefinition
= response_codes_reference_redefinition
= validity_redefinition

array_size_redefinition:
= 'REDEFINE' array_size

array_type_redefinition:
= 'REDEFINE' array_type

axis_redefinition:
= 'DELETE' 'AXIS' Identifier ';'
= 'REDEFINE' 'AXIS' Identifier '{' '}'
= 'REDEFINE' 'AXIS' Identifier '{' axis_attribute_list '}'
= 'AXIS' Identifier '{' axis_attribute_redefinition_list '}'

axis_attribute_redefinition_list:
= axis_attribute_redefinition
= axis_attribute_redefinition_list axis_attribute_redefinition

axis_attribute_redefinition:
= constant_unit_redefinition
= help_redefinition
= optional_label_redefinition
= maximum_value_redefinition
= minimum_value_redefinition
= axis_scaling_redefinition

axis_scaling_redefinition:
= 'DELETE' 'SCALING' ';'
= 'REDEFINE' axis_scaling

block_redefinition:
= 'DELETE' 'BLOCK' Identifier ';'
= 'REDEFINE' 'BLOCK' Identifier '{' '}'
= 'REDEFINE' 'BLOCK' Identifier '{' block_attribute_list '}'

```

```

= 'BLOCK' Identifier '{' block_attribute_redefinition_list '}'

block_attribute_redefinition_list:
= block_attribute_redefinition
= block_attribute_redefinition_list block_attribute_redefinition

block_attribute_redefinition:
= block_a_characteristics_redefinition
= required_label_redefinition
= block_a_parameters_redefinition
= block_a_axis_items_redefinition
= block_a_chart_items_redefinition
= block_a_collection_items_redefinition
= block_a_edit_display_items_redefinition
= block_a_file_items_redefinition
= block_a_graph_items_redefinition
= block_a_grid_items_redefinition
= help_redefinition
= block_a_image_items_redefinition
= block_a_item_array_items_redefinition
= block_a_list_items_redefinition
= block_a_local_parameters_redefinition
= block_a_menu_items_redefinition
= block_a_method_items_redefinition
= block_a_parameter_lists_redefinition
= block_a_refresh_items_redefinition
= block_a_source_items_redefinition
= block_a_unit_items_redefinition
= block_a_wao_items_redefinition
= block_a_waveform_items_redefinition
= block_a_charts_redefinition
= block_a_lists_redefinition
= block_a_graphs_redefinition
= block_a_grids_redefinition
= block_a_menus_redefinition
= block_a_methods_redefinition
= block_b_number_redefinition
= block_b_type_redefinition

block_a_characteristics_redefinition:
= 'REDEFINE' block_a_characteristics

block_a_parameters_redefinition:
= 'PARAMETERS' '{' parameter_redefinition_list '}'

parameter_redefinition_list:
= parameter_redefinition
= parameter_redefinition_list parameter_redefinition

parameter_redefinition:
= 'REDEFINE' member
= 'ADD' member

block_a_axis_items_redefinition:
= 'DELETE' 'AXIS_ITEMS' ';'
= 'REDEFINE' block_a_axis_items

block_a_chart_items_redefinition:
= 'DELETE' 'CHART_ITEMS' ';'
= 'REDEFINE' block_a_chart_items

block_a_collection_items_redefinition:
= 'DELETE' 'COLLECTION_ITEMS' ';'

```

```

= 'REDEFINE' block_a_collection_items

block_a_edit_display_items_redefinition:
= 'DELETE' 'EDIT_DISPLAY_ITEMS' ';'
= 'REDEFINE' block_a_edit_display_items

block_a_file_items_redefinition:
= 'DELETE' 'FILE_ITEMS' ';'
= 'REDEFINE' block_a_file_items

block_a_graph_items_redefinition:
= 'DELETE' 'GRAPH_ITEMS' ';'
= 'REDEFINE' block_a_graph_items

block_a_grid_items_redefinition:
= 'DELETE' 'GRID_ITEMS' ';'
= 'REDEFINE' block_a_grid_items

block_a_image_items_redefinition:
= 'DELETE' 'IMAGE_ITEMS' ';'
= 'REDEFINE' block_a_image_items

block_a_item_array_items_redefinition:
= 'DELETE' 'ITEM_ARRAY_ITEMS' ';'
= 'REDEFINE' block_a_item_array_items

block_a_list_items_redefinition:
= 'DELETE' 'LIST_ITEMS' ';'
= 'REDEFINE' block_a_list_items

block_a_local_parameters_redefinition:
= 'LOCAL_PARAMETERS' '{' parameter_redefinition_list '}'

block_a_menu_items_redefinition:
= 'DELETE' 'MENU_ITEMS' ';'
= 'REDEFINE' block_a_menu_items

block_a_method_items_redefinition:
= 'DELETE' 'METHOD_ITEMS' ';'
= 'REDEFINE' block_a_method_items

block_a_parameter_lists_redefinition:
= 'PARAMETER_LISTS' '{' parameter_redefinition_list '}'

block_a_refresh_items_redefinition:
= 'DELETE' 'REFRESH_ITEMS' ';'
= 'REDEFINE' block_a_refresh_items

block_a_source_items_redefinition:
= 'DELETE' 'SOURCE_ITEMS' ';'
= 'REDEFINE' block_a_source_items

block_a_unit_items_redefinition:
= 'DELETE' 'UNIT_ITEMS' ';'
= 'REDEFINE' block_a_unit_items

block_a_wao_items_redefinition:
= 'DELETE' 'WRITE_AS_ONE_ITEMS' ';'
= 'REDEFINE' block_a_wao_items

block_a_waveform_items_redefinition:
= 'DELETE' 'WAVEFORM_ITEMS' ';'
= 'REDEFINE' block_a_waveform_items

```



```

block_a_charts_redefinition:
  = 'DELETE' 'CHARTS' ';';
  = 'REDEFINE' block_a_charts
  = 'CHARTS' '{' block_a_member_redefinition_list '}'

block_a_lists_redefinition:
  = 'DELETE' 'LISTS' ';';
  = 'REDEFINE' block_a_lists
  = 'LISTS' '{' block_a_member_redefinition_list '}'

block_a_graphs_redefinition:
  = 'DELETE' 'GRAPHS' ';';
  = 'REDEFINE' block_a_graphs
  = 'GRAPHS' '{' block_a_member_redefinition_list '}'

block_a_grids_redefinition:
  = 'DELETE' 'GRIDS' ';';
  = 'REDEFINE' block_a_grids
  = 'GRIDS' '{' block_a_member_redefinition_list '}'

block_a_menus_redefinition:
  = 'DELETE' 'MENUS' ';';
  = 'REDEFINE' block_a_menus
  = 'MENUS' '{' block_a_member_redefinition_list '}'

block_a_methods_redefinition:
  = 'DELETE' 'METHODS' ';';
  = 'REDEFINE' block_a_methods
  = 'METHODS' '{' block_a_member_redefinition_list '}'

block_a_member_redefinition_list:
  = block_a_member_redefinition
  = block_a_member_redefinition_list block_a_member_redefinition

block_a_member_redefinition:
  = 'DELETE' Identifier ';';
  = 'REDEFINE' member
  = 'ADD' member

block_b_type_redefinition:
  = 'REDEFINE' block_b_type

block_b_number_redefinition:
  = 'REDEFINE' block_b_number

chart_redefinition:
  = 'DELETE' 'CHART' Identifier ';';
  = 'REDEFINE' 'CHART' Identifier '{' chart_attribute_list '}'
  = 'CHART' Identifier '{' chart_attribute_redefinition_list '}'

chart_attribute_redefinition_list:
  = chart_attribute_redefinition
  = chart_attribute_redefinition_list chart_attribute_redefinition

chart_attribute_redefinition:
  = members_redefinition
  = chart_cycle_time_redefinition
  = height_redefinition
  = help_redefinition
  = optional_label_redefinition
  = chart_length_redefinition
  = chart_type_redefinition

```

```

= validity_redefinition
= width_redefinition

chart_cycle_time_redefinition:
= 'DELETE' 'CYCLE_TIME' ';'
= 'REDEFINE' chart_cycle_time

chart_length_redefinition:
= 'DELETE' 'LENGTH' ';'
= 'REDEFINE' chart_length

chart_type_redefinition:
= 'DELETE' 'TYPE' ';'
= 'REDEFINE' chart_type

collection_redefinition:
= 'DELETE' 'COLLECTION' Identifier ';'
= 'REDEFINE' 'COLLECTION' Identifier '{' '}'
= 'REDEFINE' 'COLLECTION' 'OF' item_type Identifier '{' '}'
= 'REDEFINE' 'COLLECTION' Identifier '{' collection_attribute_list '}'
= 'REDEFINE' 'COLLECTION' 'OF' item_type Identifier '{'
  collection_attribute_list '}'
= 'COLLECTION' Identifier '{'
  collection_attribute_redefinition_list '}'
= 'COLLECTION' 'OF' item_type Identifier '{'
  collection_attribute_redefinition_list '}'

collection_attribute_redefinition_list:
= collection_attribute_redefinition
= collection_attribute_redefinition_list
  collection_attribute_redefinition

collection_attribute_redefinition:
= members_redefinition
= help_redefinition
= optional_label_redefinition
= validity_redefinition

command_redefinition:
= 'DELETE' 'COMMAND' Identifier ';'
= 'REDEFINE' 'COMMAND' Identifier '{' '}'
= 'REDEFINE' 'COMMAND' Identifier '{' command_attribute_list '}'
= 'COMMAND' Identifier '{' command_attribute_redefinition_list '}'

command_attribute_redefinition_list:
= command_attribute_redefinition
= command_attribute_redefinition_list command_attribute_redefinition

command_attribute_redefinition:
= command_transaction_redefinition
= command_index_redefinition
= command_slot_redefinition
= command_sub_slot_redefinition
= command_connection_redefinition
= command_header_redefinition
= command_module_redefinition
= response_codes_reference_redefinition

command_transaction_redefinition:
= 'ADD' command_transaction
= 'REDEFINE' command_transaction
= 'DELETE' 'TRANSACTION' ';'
= 'DELETE' 'TRANSACTION' Integer ';'

```

```
command_index_redefinition:
  = 'DELETE' 'INDEX' ';'
  = 'REDEFINE' command_index

command_slot_redefinition:
  = 'DELETE' 'SLOT' ';'
  = 'REDEFINE' command_slot

command_sub_slot_redefinition:
  = 'DELETE' 'SUB_SLOT' ';'
  = 'REDEFINE' command_sub_slot

command_connection_redefinition:
  = 'DELETE' 'CONNECTION' ';'
  = 'REDEFINE' command_connection

command_header_redefinition:
  = 'DELETE' 'HEADER' ';'
  = 'REDEFINE' command_header

command_module_redefinition:
  = 'DELETE' 'MODULE' ';'
  = 'REDEFINE' command_module

component_redefinition:
  = 'DELETE' 'COMPONENT' Identifier ';'
  = 'REDEFINE' 'COMPONENT' Identifier '{' component_attribute_list '}'
  = 'COMPONENT' Identifier '{' component_attribute_redefinition_list '}'

component_attribute_redefinition_list:
  = component_attribute_redefinition
  = component_attribute_redefinition_list
    component_attribute_redefinition

component_attribute_redefinition:
  = required_label_redefinition
  = can_delete_redefinition
  = check_configuration_redefinition
  = classification_redefinition
  = component_parent_redefinition
  = component_path_redefinition
  = component_relations_redefinition
  = declaration_redefinition
  = detect_method_redefinition
  = edd_redefinition
  = help_redefinition
  = initial_values_redefinition
  = protocol_redefinition
  = redundancy_redefinition
  = scan_method_redefinition
  = scan_list_redefinition
  = supplied_interface_redefinition

can_delete_redefinition:
  = 'DELETE' 'CAN_DELETE' ';'
  = 'REDEFINE' can_delete

check_configuration_redefinition:
  = 'DELETE' 'CHECK_CONFIGURATION' ';'
  = 'REDEFINE' check_configuration

classification_redefinition:
```

```

= 'DELETE' 'CLASSIFICATION' ';'
= 'REDEFINE' classification

component_parent_redefinition:
= 'DELETE' 'COMPONENT_PARENT' ';'
= 'REDEFINE' component_parent

component_path_redefinition:
= 'DELETE' 'COMPONENT_PATH' ';'
= 'REDEFINE' component_path

component_relations_redefinition:
= 'DELETE' 'COMPONENT_REDEFINITIONS' ';'
= 'REDEFINE' component_relations

declaration_redefinition:
= 'DELETE' 'DECLARATION' ';'
= 'REDEFINE' declaration

detect_method_redefinition:
= 'DELETE' 'DETECT' ';'
= 'REDEFINE' detect_method

edd_redefinition:
= 'DELETE' 'EDD' ';'
= 'REDEFINE' edd

initial_values_redefinition:
= 'DELETE' 'INITIAL_VALUES' ';'
= 'REDEFINE' initial_values

protocol_redefinition:
= 'DELETE' 'PROTOCOL' ';'
= 'REDEFINE' protocol

redundancy_redefinition:
= 'DELETE' 'REDUNDANCY' ';'
= 'REDEFINE' redundancy

scan_method_redefinition:
= 'DELETE' 'SCAN' ';'
= 'REDEFINE' scan_method

scan_list_redefinition:
= 'DELETE' 'SCAN_LIST' ';'
= 'REDEFINE' scan_list

supplied_interface_redefinition:
= 'DELETE' 'SUPPLIED_INTERFACE' ';'
= 'REDEFINE' supplied_interface

component_folder_redefinition:
= 'DELETE' 'COMPONENT_FOLDER' Identifier ';'
= 'REDEFINE' 'COMPONENT_FOLDER' Identifier '{'
    component_folder_attribute_list '}'
= 'COMPONENT_FOLDER' Identifier '{'
    component_folder_attribute_redefinition_list '}'

component_folder_attribute_redefinition_list:
= component_folder_attribute_redefinition
= component_folder_attribute_redefinition_list
    component_folder_attribute_redefinition

```

```

component_folder_attribute_redefinition:
= required_label_redefinition
= classification_redefinition
= component_parent_redefinition
= component_path_redefinition
= help_redefinition
= protocol_redefinition

component_reference_redefinition:
= 'DELETE' 'COMPONENT_REFERENCE' Identifier ';'
= 'REDEFINE' 'COMPONENT_REFERENCE' Identifier '{'
  component_reference_attribute_list '}'
= 'COMPONENT_REFERENCE' Identifier '{'
  component_reference_attribute_redefinition_list '}'

component_reference_attribute_redefinition_list:
= component_reference_attribute_redefinition
= component_reference_attribute_redefinition_list
  component_reference_attribute_redefinition

component_reference_attribute_redefinition:
= classification_redefinition
= component_parent_redefinition
= component_path_redefinition
= device_revision_attribute_redefinition
= device_type_attribute_redefinition
= manufacturer_attribute_redefinition
= protocol_redefinition

device_revision_attribute_redefinition:
= 'DELETE' 'DEVICE_REVISION' ';'
= 'REDEFINE' device_revision_attribute

device_type_attribute_redefinition:
= 'DELETE' 'DEVICE_TYPE' ';'
= 'REDEFINE' device_type_attribute

manufacturer_attribute_redefinition:
= 'DELETE' 'MANUFACTURER' ';'
= 'REDEFINE' manufacturer_attribute

component_relation_redefinition:
= 'DELETE' 'COMPONENT_RELATION' Identifier ';'
= 'REDEFINE' 'COMPONENT_RELATION' Identifier '{'
  component_relation_attribute_list '}'
= 'COMPONENT_RELATION' Identifier '{'
  component_relation_attribute_redefinition_list '}'

component_relation_attribute_redefinition_list:
= component_relation_attribute_redefinition
= component_relation_attribute_redefinition_list
  component_relation_attribute_redefinition

component_relation_attribute_redefinition:
= components_redefinition
= required_label_redefinition
= relation_type_redefinition
= addressing_redefinition
= help_redefinition
= maximum_number_redefinition
= minimum_number_redefinition
= required_interface_redefinition
= supplied_interface_redefinition

```

```

components_redefinition:
  = 'REDEFINE' components
  = 'COMPONENTS' '{' component_relation_reference_redefinition_list '}'

component_relation_reference_redefinition_list:
  = component_relation_reference_redefinition
  = component_relation_reference_redefinition_list ','
  component_relation_reference_redefinition

component_relation_reference_redefinition:
  = 'DELETE' Identifier ';'
  = 'REDEFINE' component_relation_reference_specifier
  = 'ADD' component_relation_reference_specifier
  = Identifier '{' component_relation_arg_redefinition_list '}'

component_relation_arg_redefinition_list:
  = component_relation_arg_redefinition
  = component_relation_arg_redefinition_list
  component_relation_arg_redefinition

component_relation_arg_redefinition:
  = minimum_number_redefinition
  = maximum_number_redefinition
  = auto_create_redefinition
  = filter_redefinition
  = required_ranges_redefinition

auto_create_redefinition:
  = 'DELETE' 'AUTO_CREATE' ';'
  = 'REDEFINE' auto_create

filter_redefinition:
  = 'DELETE' 'FILTER' ';'
  = 'REDEFINE' filter

required_ranges_redefinition:
  = 'DELETE' 'REQUIRED_RANGES' ';'
  = 'REDEFINE' required_ranges

relation_type_redefinition:
  = 'REDEFINE' relation_type

addressing_redefinition:
  = 'DELETE' 'ADDRESSING' ';'
  = 'REDEFINE' addressing

maximum_number_redefinition:
  = 'DELETE' 'MAXIMUM_NUMBER' ';'
  = 'REDEFINE' maximum_number

minimum_number_redefinition:
  = 'DELETE' 'MINIMUM_NUMBER' ';'
  = 'REDEFINE' minimum_number

required_interface_redefinition:
  = 'DELETE' 'REQUIRED_INTERFACE' ';'
  = 'REDEFINE' required_interface

connection_redefinition:
  = 'DELETE' 'CONNECTION' Identifier ';'
  = 'REDEFINE' 'CONNECTION' Identifier '{' '}'
  = 'REDEFINE' 'CONNECTION' Identifier '{'

```

```

    connection_attribute_list '}'
    = 'CONNECTION' Identifier '{'
    connection_attribute_redefinition_list '}'

connection_attribute_redefinition_list:
    = connection_attribute_redefinition
    = connection_attribute_redefinition_list
    connection_attribute_redefinition

connection_attribute_redefinition:
    = appinstance_redefinition

appinstance_redefinition:
    = 'REDEFINE' appinstance

domain_redefinition:
    = 'DELETE' 'DOMAIN' Identifier ';'
    = 'REDEFINE' 'DOMAIN' Identifier '{' domain_attribute_list '}'
    = 'DOMAIN' Identifier '{' domain_attribute_redefinition_list '}'

domain_attribute_redefinition_list:
    = domain_attribute_redefinition
    = domain_attribute_redefinition_list domain_attribute_redefinition

domain_attribute_redefinition:
    = handling_redefinition
    = response_codes_reference_redefinition

edit_display_redefinition:
    = 'DELETE' 'EDIT_DISPLAY' Identifier ';'
    = 'REDEFINE' 'EDIT_DISPLAY' Identifier '{' '}'
    = 'REDEFINE' 'EDIT_DISPLAY' Identifier '{'
    edit_display_attribute_list '}'
    = 'EDIT_DISPLAY' Identifier '{'
    edit_display_attribute_redefinition_list '}'

edit_display_attribute_redefinition_list:
    = edit_display_attribute_redefinition
    = edit_display_attribute_redefinition_list
    edit_display_attribute_redefinition

edit_display_attribute_redefinition:
    = edit_items_redefinition
    = required_label_redefinition
    = display_items_redefinition
    = post_edit_actions_redefinition
    = pre_edit_actions_redefinition

edit_items_redefinition:
    = 'REDEFINE' edit_items

display_items_redefinition:
    = 'DELETE' 'DISPLAY_ITEMS' ';'
    = 'REDEFINE' display_items

file_redefinition:
    = 'DELETE' 'FILE' Identifier ';'
    = 'REDEFINE' 'FILE' Identifier '{' file_attribute_list '}'
    = 'FILE' Identifier '{' file_attribute_redefinition_list '}'

file_attribute_redefinition_list:
    = file_attribute_redefinition
    = file_attribute_redefinition_list file_attribute_redefinition

```

```

file_attribute_redefinition:
  = members_redefinition
  = optional_label_redefinition
  = help_redefinition

graph_redefinition:
  = 'DELETE' 'GRAPH' Identifier ';'
  = 'REDEFINE' 'GRAPH' Identifier '{' graph_attribute_list '}'
  = 'GRAPH' Identifier '{' graph_attribute_redefinition_list '}'

graph_attribute_redefinition_list:
  = graph_attribute_redefinition
  = graph_attribute_redefinition_list graph_attribute_redefinition

graph_attribute_redefinition:
  = members_redefinition
  = graph_cycle_time_redefinition
  = help_redefinition
  = optional_label_redefinition
  = height_redefinition
  = validity_redefinition
  = width_redefinition
  = graph_x_axis_redefinition

graph_cycle_time_redefinition:
  = 'DELETE' 'CYCLE_TIME' ';'
  = 'REDEFINE' graph_cycle_time

graph_x_axis_redefinition:
  = 'DELETE' 'X_AXIS' ';'
  = 'REDEFINE' graph_x_axis

grid_redefinition:
  = 'DELETE' 'GRID' Identifier ';'
  = 'REDEFINE' 'GRID' Identifier '{' grid_attribute_list '}'
  = 'GRID' Identifier '{' grid_attribute_redefinition_list '}'

grid_attribute_redefinition_list:
  = grid_attribute_redefinition
  = grid_attribute_redefinition_list grid_attribute_redefinition

grid_attribute_redefinition:
  = grid_vectors_redefinition
  = handling_redefinition
  = height_redefinition
  = help_redefinition
  = optional_label_redefinition
  = grid_orientation_redefinition
  = validity_redefinition
  = width_redefinition

grid_vectors_redefinition:
  = 'REDEFINE' grid_vectors

grid_orientation_redefinition:
  = 'DELETE' 'ORIENTATION' ';'
  = 'REDEFINE' grid_orientation

image_redefinition:
  = 'DELETE' 'IMAGE' Identifier ';'
  = 'REDEFINE' 'IMAGE' Identifier '{' image_attribute_list '}'
  = 'IMAGE' Identifier '{' image_attribute_redefinition_list '}'

```



```

image_attribute_redefinition_list:
  = image_attribute_redefinition
  = image_attribute_redefinition_list image_attribute_redefinition

image_attribute_redefinition:
  = image_path_redefinition
  = optional_label_redefinition
  = help_redefinition
  = image_link_redefinition
  = validity_redefinition

image_path_redefinition:
  = 'REDEFINE' image_path

image_link_redefinition:
  = 'DELETE' 'LINK' ';'
  = 'REDEFINE' image_link

interface_redefinition:
  = 'DELETE' 'INTERFACE' Identifier ';'
  = 'REDEFINE' 'INTERFACE' Identifier '{' interface_attribute_list '}'
  = 'INTERFACE' Identifier '{' interface_attribute_redefinition_list '}'

interface_attribute_redefinition_list:
  = interface_attribute_redefinition
  = interface_attribute_redefinition_list
    interface_attribute_redefinition

interface_attribute_redefinition:
  = required_declaration_redefinition
  = required_label_redefinition
  = help_redefinition

required_declaration_redefinition:
  = 'REDEFINE' declaration

item_array_redefinition:
  = 'DELETE' 'ITEM_ARRAY' Identifier ';'
  = 'REDEFINE' 'ITEM_ARRAY' 'OF' item_type Identifier '{' '}'
  = 'REDEFINE' 'ITEM_ARRAY' 'OF' item_type Identifier '{'
    item_array_attribute_list '}'
  = 'ITEM_ARRAY' 'OF' item_type Identifier '{'
    item_array_attribute_redefinition_list '}'

item_array_attribute_redefinition_list:
  = item_array_attribute_redefinition
  = item_array_attribute_redefinition_list
    item_array_attribute_redefinition

item_array_attribute_redefinition:
  = elements_redefinition
  = help_redefinition
  = optional_label_redefinition
  = validity_redefinition

elements_redefinition:
  = 'ELEMENTS' '{' element_redefinition_list '}'
  = 'REDEFINE' elements

element_redefinition_list:
  = element_redefinition
  = element_redefinition_list element_redefinition

```

```

element_redefinition:
= 'DELETE' Integer ';'
= 'REDEFINE' element
= 'ADD' element

list_redefinition:
= 'DELETE' 'LIST' Identifier ';'
= 'REDEFINE' 'LIST' Identifier '{' list_attribute_list '}'
= 'LIST' Identifier '{' list_attribute_redefinition_list '}'

list_attribute_redefinition_list:
= list_attribute_redefinition
= list_attribute_redefinition_list list_attribute_redefinition

list_attribute_redefinition:
= list_type_redefinition
= list_capacity_redefinition
= list_count_redefinition
= help_redefinition
= optional_label_redefinition
= validity_redefinition

list_type_redefinition:
= 'REDEFINE' list_type

list_capacity_redefinition:
= 'DELETE' 'CAPACITY' ';'
= 'REDEFINE' list_capacity

list_count_redefinition:
= 'DELETE' 'COUNT' ';'
= 'REDEFINE' list_count

menu_redefinition:
= 'DELETE' 'MENU' Identifier ';'
= 'REDEFINE' 'MENU' Identifier '{' '}'
= 'REDEFINE' 'MENU' Identifier '{' menu_attribute_list '}'
= 'MENU' Identifier '{' menu_attribute_redefinition_list '}'

menu_attribute_redefinition_list:
= menu_attribute_redefinition
= menu_attribute_redefinition_list menu_attribute_redefinition

menu_attribute_redefinition:
= menu_items_redefinition
= required_label_redefinition
= menu_access_redefinition
= help_redefinition
= post_edit_actions_redefinition
= post_read_actions_redefinition
= post_write_actions_redefinition
= pre_edit_actions_redefinition
= pre_read_actions_redefinition
= pre_write_actions_redefinition
= menu_purpose_redefinition
= menu_role_redefinition
= menu_style_redefinition
= validity_redefinition

menu_items_redefinition:
= 'REDEFINE' menu_items

```

```

menu_access_redefinition:
  = 'DELETE' 'ACCESS' ';'
  = 'REDEFINE' menu_access

menu_purpose_redefinition:
  = 'DELETE' 'PURPOSE' ';'
  = 'REDEFINE' menu_purpose

menu_role_redefinition:
  = 'DELETE' 'ROLE' ';'
  = 'REDEFINE' menu_role

menu_style_redefinition:
  = 'DELETE' 'STYLE' ';'
  = 'REDEFINE' menu_style

method_redefinition:
  = 'DELETE' 'METHOD' Identifier ';'
  = 'REDEFINE' 'METHOD' Identifier '{' '}'
  = 'REDEFINE' 'METHOD' Identifier '{' method_attribute_list '}'
  = 'METHOD' Identifier '{' method_attribute_redefinition_list '}'

method_attribute_redefinition_list:
  = method_attribute_redefinition
  = method_attribute_redefinition_list method_attribute_redefinition

method_attribute_redefinition:
  = method_definition_redefinition
  = method_access_redefinition
  = variable_class_redefinition
  = help_redefinition
  = optional_label_redefinition
  = validity_redefinition

method_definition_redefinition:
  = 'REDEFINE' method_definition

method_access_redefinition:
  = 'DELETE' 'ACCESS' ';'
  = 'REDEFINE' method_access

program_redefinition:
  = 'DELETE' 'PROGRAM' Identifier ';'
  = 'REDEFINE' 'PROGRAM' Identifier '{' '}'
  = 'REDEFINE' 'PROGRAM' Identifier '{' program_attribute_list '}'
  = 'PROGRAM' Identifier '{' program_attribute_redefinition_list '}'

program_attribute_redefinition_list:
  = program_attribute_redefinition
  = program_attribute_redefinition_list program_attribute_redefinition

program_attribute_redefinition:
  = arguments_redefinition
  = response_codes_reference_redefinition

arguments_redefinition:
  = 'DELETE' 'ARGUMENTS' ';'
  = 'REDEFINE' arguments

record_redefinition:
  = 'DELETE' 'RECORD' Identifier ';'
  = 'REDEFINE' 'RECORD' Identifier '{' record_attribute_list '}'
  = 'RECORD' Identifier '{' record_attribute_redefinition_list '}'

```

```

record_attribute_redefinition_list:
  = record_attribute_redefinition
  = record_attribute_redefinition_list record_attribute_redefinition

record_attribute_redefinition:
  = required_label_redefinition
  = members_redefinition
  = help_redefinition
  = response_codes_reference_redefinition
  = validity_redefinition

refresh_relation_redefinition:
  = 'DELETE' 'REFRESH' Identifier ';'
  = 'REDEFINE' 'REFRESH' Identifier '{' '}'
  = 'REDEFINE' 'REFRESH' Identifier
    '{' refresh_relation_left ':' refresh_relation_right '}'

unit_relation_redefinition:
  = 'DELETE' 'UNIT' Identifier ';'
  = 'REDEFINE' 'UNIT' Identifier '{' '}'
  = 'REDEFINE' 'UNIT' Identifier
    '{' unit_relation_left ':' unit_relation_right '}'

wao_relation_redefinition:
  = 'DELETE' 'WRITE_AS_ONE' Identifier ';'
  = 'REDEFINE' 'WRITE_AS_ONE' Identifier '{' '}'
  = 'REDEFINE' 'WRITE_AS_ONE' Identifier
    '{' wao_specifier '}'

response_codes_definition_redefinition:
  = 'DELETE' 'RESPONSE_CODES' Identifier ';'
  = 'REDEFINE' 'RESPONSE_CODES' Identifier '{' '}'
  = 'REDEFINE' 'RESPONSE_CODES' Identifier '{'
    response_codes_specifier_list '}'
  = 'RESPONSE_CODES' Identifier '{' response_code_redefinition_list
    '}'

response_code_redefinition_list:
  = response_code_redefinition
  = response_code_redefinition_list response_code_redefinition

response_code_redefinition:
  = 'DELETE' Integer ';'
  = 'REDEFINE' response_code
  = 'ADD' response_code

emphasis_redefinition:
  = 'DELETE' 'EMPHASIS' ';'
  = 'REDEFINE' emphasis

height_redefinition:
  = 'DELETE' 'HEIGHT' ';'
  = 'REDEFINE' height

help_redefinition:
  = 'DELETE' 'HELP' ';'
  = 'REDEFINE' help

required_label_redefinition:
  = 'REDEFINE' label

optional_label_redefinition:

```

```
= 'DELETE' 'LABEL' ';'
= 'REDEFINE' label

line_color_redefinition:
= 'DELETE' 'LINE_COLOR' ';'
= 'REDEFINE' line_color

line_type_redefinition:
= 'DELETE' 'LINE_TYPE' ';'
= 'REDEFINE' line_type

response_codes_reference_redefinition:
= 'DELETE' 'RESPONSE_CODES' ';'
= 'REDEFINE' response_codes

validity_redefinition:
= 'DELETE' 'VALIDITY' ';'
= 'REDEFINE' validity

width_redefinition:
= 'DELETE' 'WIDTH' ';'
= 'REDEFINE' width

members_redefinition:
= 'MEMBERS' '{' member_redefinition_list '}'
= 'REDEFINE' members

member_redefinition_list:
= member_redefinition
= member_redefinition_list member_redefinition

member_redefinition:
= 'DELETE' Identifier ';'
= 'REDEFINE' member
= 'ADD' member

source_redefinition:
= 'DELETE' 'SOURCE' Identifier ';'
= 'REDEFINE' 'SOURCE' Identifier '{' source_attribute_list '}'
= 'SOURCE' Identifier '{' source_attribute_redefinition_list '}'

source_attribute_redefinition_list:
= source_attribute_redefinition
= source_attribute_redefinition_list source_attribute_redefinition

source_attribute_redefinition:
= members_redefinition
= emphasis_redefinition
= source_exit_actions_redefinition
= help_redefinition
= source_init_actions_redefinition
= optional_label_redefinition
= line_color_redefinition
= line_type_redefinition
= source_refresh_actions_redefinition
= validity_redefinition
= source_y_axis_redefinition

source_exit_actions_redefinition:
= 'DELETE' 'REFRESH_ACTIONS' ';'
= 'REDEFINE' source_exit_actions

source_init_actions_redefinition:
```

```

= 'DELETE' 'INIT_ACTIONS' ';'
= 'REDEFINE' source_init_actions

source_refresh_actions_redefinition:
= 'DELETE' 'REFRESH_ACTIONS' ';'
= 'REDEFINE' source_refresh_actions

source_y_axis_redefinition:
= 'DELETE' 'Y_AXIS' ';'
= 'REDEFINE' waveform_y_axis

template_redefinition:
= 'DELETE' 'TEMPLATE' Identifier ';'
= 'REDEFINE' 'TEMPLATE' Identifier '{' template_attribute_list '}'
= 'TEMPLATE' Identifier '{' template_attribute_redefinition_list '}'

template_attribute_redefinition_list:
= template_attribute_redefinition
= template_attribute_redefinition_list template_attribute_redefinition

template_attribute_redefinition:
= template_default_values_redefinition
= required_label_redefinition
= help_redefinition

template_default_values_redefinition:
= 'REDEFINE' template_default_values

variable_redefinition:
= 'DELETE' 'VARIABLE' Identifier ';'
= 'REDEFINE' 'VARIABLE' Identifier '{' variable_attribute_list
  '}'
= 'VARIABLE' Identifier '{' variable_attribute_redefinition_list
  '}'

variable_attribute_redefinition_list:
= variable_attribute_redefinition
= variable_attribute_redefinition_list variable_attribute_redefinition

variable_attribute_redefinition:
= variable_class_redefinition
= type_redefinition
= constant_unit_redefinition
= default_value_redefinition
= handling_redefinition
= help_redefinition
= initial_value_redefinition
= optional_label_redefinition
= post_edit_actions_redefinition
= post_read_actions_redefinition
= post_write_actions_redefinition
= pre_edit_actions_redefinition
= pre_read_actions_redefinition
= pre_write_actions_redefinition
= read_timeout_redefinition
= refresh_actions_redefinition
= response_codes_reference_redefinition
= variable_style_redefinition
= validity_redefinition
= write_timeout_redefinition

variable_class_redefinition:
= 'REDEFINE' variable_class

```

```

type_redefinition:
= 'REDEFINE' 'TYPE' arithmetic_type
= 'TYPE' arithmetic_type_redefinition
= 'REDEFINE' 'TYPE' enumerated_type
= 'TYPE' enumerated_type_redefinition
= 'REDEFINE' 'TYPE' string_type
= 'TYPE' string_type_redefinition
= 'REDEFINE' 'TYPE' date_time_type
= 'TYPE' date_time_type_redefinition

arithmetic_type_redefinition:
= 'INTEGER' arithmetic_size '{'
  arithmetic_option_redefinition_list '}'
= 'UNSIGNED_INTEGER' arithmetic_size '{'
  arithmetic_option_redefinition_list '}'
= 'FLOAT' '{' arithmetic_option_redefinition_list '}'
= 'DOUBLE' '{' arithmetic_option_redefinition_list '}'

arithmetic_option_redefinition_list:
= arithmetic_option_redefinition
= arithmetic_option_redefinition_list arithmetic_option_redefinition

arithmetic_option_redefinition:
= default_value_redefinition
= initial_value_redefinition
= display_format_redefinition
= edit_format_redefinition
= arithmetic_enumerator_list_redefinition
= maximum_value_redefinition
= maximum_value_n_redefinition
= minimum_value_redefinition
= minimum_value_n_redefinition
= scaling_factor_redefinition

display_format_redefinition:
= 'DELETE' 'DISPLAY_FORMAT' ';'
= 'REDEFINE' display_format

edit_format_redefinition:
= 'DELETE' 'EDIT_FORMAT' ';'
= 'REDEFINE' edit_format

arithmetic_enumerator_list_redefinition:
= 'DELETE' Integer ';'
= 'DELETE' Real ';'
= 'REDEFINE' arithmetic_enumerator
= 'ADD' arithmetic_enumerator

maximum_value_redefinition:
= 'DELETE' 'MAX_VALUE' ';'
= 'REDEFINE' maximum_value

maximum_value_n_redefinition:
= 'DELETE' 'MAX_VALUE1' ';'
= 'DELETE' 'MAX_VALUE2' ';'
= 'DELETE' 'MAX_VALUE3' ';'
= 'DELETE' 'MAX_VALUE4' ';'
= 'DELETE' 'MAX_VALUE5' ';'
= 'DELETE' 'MAX_VALUE6' ';'
= 'DELETE' 'MAX_VALUE7' ';'
= 'DELETE' 'MAX_VALUE8' ';'
= 'DELETE' 'MAX_VALUE9' ';'

```

```

= 'REDEFINE' maximum_value_n

minimum_value_redefinition:
= 'DELETE' 'MIN_VALUE' ';'
= 'REDEFINE' minimum_value

minimum_value_n_redefinition:
= 'DELETE' 'MIN_VALUE1' ';'
= 'DELETE' 'MIN_VALUE2' ';'
= 'DELETE' 'MIN_VALUE3' ';'
= 'DELETE' 'MIN_VALUE4' ';'
= 'DELETE' 'MIN_VALUE5' ';'
= 'DELETE' 'MIN_VALUE6' ';'
= 'DELETE' 'MIN_VALUE7' ';'
= 'DELETE' 'MIN_VALUE8' ';'
= 'DELETE' 'MIN_VALUE8' ';'
= 'REDEFINE' minimum_value_n

scaling_factor_redefinition:
= 'DELETE' 'SCALING_FACTOR' ';'
= 'REDEFINE' scaling_factor

enumerated_type_redefinition:
= 'ENUMERATED' enumerated_size '{'
    enumerated_option_redefinition_list '}'
= 'BIT_ENUMERATED' enumerated_size '{'
    bit_enumerated_option_redefinition_list '}'

enumerated_option_redefinition_list:
= enumerated_option_redefinition
= enumerated_option_redefinition_list enumerated_option_redefinition

enumerated_option_redefinition:
= default_value_redefinition
= initial_value_redefinition
= integral_enumerator_redefinition

integral_enumerator_redefinition:
= 'DELETE' Integer ';'
= 'REDEFINE' integral_enumerator
= 'ADD' integral_enumerator

bit_enumerated_option_redefinition_list:
= bit_enumerated_option_redefinition
= bit_enumerated_option_redefinition_list

bit_enumerated_option_redefinition

bit_enumerated_option_redefinition:
= default_value_redefinition
= initial_value_redefinition
= bit_enumerators_redefinition

bit_enumerators_redefinition:
= 'DELETE' Integer ';'
= 'REDEFINE' bit_enumerator
= 'ADD' bit_enumerator

string_type_redefinition:
= 'ASCII' string_size '{' string_option_redefinition_list '}'
= 'BITSTRING' string_size '{' string_option_redefinition_list '}'
= 'EUC' string_size '{' string_option_redefinition_list '}'
= 'OCTET' string_size '{' string_option_redefinition_list '}'
= 'PACKED_ASCII' string_size '{' string_option_redefinition_list '}'

```



```

= 'PASSWORD' string_size '{' string_option_redefinition_list '}'
= 'VISIBLE' string_size '{' string_option_redefinition_list '}'

string_option_redefinition_list:
= string_option_redefinition
= string_option_redefinition_list string_option_redefinition

string_option_redefinition:
= string_default_value_redefinition
= string_initial_value_redefinition
= string_enumerator_redefinition

string_default_value_redefinition:
= 'DELETE' 'DEFAULT_VALUE' ';'
= 'REDEFINE' string_default_value

string_initial_value_redefinition:
= 'DELETE' 'INITIAL_VALUE' ';'
= 'REDEFINE' string_initial_value

string_enumerator_redefinition:
= 'DELETE' string ';'
= 'REDEFINE' string_enumerator
= 'ADD' string_enumerator

date_time_type_redefinition:
= 'DATE' '{' date_time_option_redefinition_list '}'
= 'DATE_AND_TIME' '{' date_time_option_redefinition_list '}'
= 'DURATION' '{' date_time_option_redefinition_list '}'
= 'TIME' '{' date_time_option_redefinition_list '}'
= 'TIME_VALUE' date_time_size '{'
= time_value_option_redefinition_list '}'

date_time_option_redefinition_list:
= date_time_option_redefinition
= date_time_option_redefinition_list date_time_option_redefinition

date_time_option_redefinition:
= default_value_redefinition
= initial_value_redefinition

time_value_option_redefinition_list:
= time_value_option_redefinition
= time_value_option_redefinition_list time_value_option_redefinition

time_value_option_redefinition:
= default_value_redefinition
= display_format_redefinition
= edit_format_redefinition
= initial_value_redefinition
= time_format_redefinition
= time_scale_redefinition

time_format_redefinition:
= 'DELETE' 'TIME_FORMAT' ';'
= 'REDEFINE' time_format

time_scale_redefinition:
= 'DELETE' 'TIME_SCALE' ';'
= 'REDEFINE' time_scale

constant_unit_redefinition:
= 'DELETE' 'CONSTANT_UNIT' ';'

```

```

= 'REDEFINE' constant_unit

default_value_redefinition:
= 'DELETE' 'DEFAULT_VALUE' ';'
= 'REDEFINE' default_value

handling_redefinition:
= 'DELETE' 'HANDLING' ';'
= 'REDEFINE' handling

initial_value_redefinition:
= 'DELETE' 'INITIAL_VALUE' ';'
= 'REDEFINE' initial_value

post_edit_actions_redefinition:
= 'DELETE' 'POST_EDIT_ACTIONS' ';'
= 'REDEFINE' post_edit_actions

post_read_actions_redefinition:
= 'DELETE' 'POST_READ_ACTIONS' ';'
= 'REDEFINE' post_read_actions

post_write_actions_redefinition:
= 'DELETE' 'POST_WRITE_ACTIONS' ';'
= 'REDEFINE' post_write_actions

pre_edit_actions_redefinition:
= 'DELETE' 'PRE_EDIT_ACTIONS' ';'
= 'REDEFINE' pre_edit_actions

pre_read_actions_redefinition:
= 'DELETE' 'PRE_READ_ACTIONS' ';'
= 'REDEFINE' pre_read_actions

pre_write_actions_redefinition:
= 'DELETE' 'PRE_WRITE_ACTIONS' ';'
= 'REDEFINE' pre_write_actions

read_timeout_redefinition:
= 'DELETE' 'READ_TIMEOUT' ';'
= 'REDEFINE' read_timeout

refresh_actions_redefinition:
= 'DELETE' 'REFRESH_ACTIONS' ';'
= 'REDEFINE' refresh_actions

variable_style_redefinition:
= 'DELETE' 'STYLE' ';'
= 'REDEFINE' variable_style

write_timeout_redefinition:
= 'DELETE' 'WRITE_TIMEOUT' ';'
= 'REDEFINE' write_timeout

variable_list_redefinition:
= 'DELETE' 'VARIABLE_LIST' Identifier ';'
= 'REDEFINE' 'VARIABLE_LIST' Identifier '{' '}'
= 'REDEFINE' 'VARIABLE_LIST' Identifier '{'
    variable_list_attribute_list '}'
= 'VARIABLE_LIST' Identifier '{'
    variable_list_attribute_redefinition_list '}'

variable_list_attribute_redefinition_list:

```

```

= variable_list_attribute_redefinition
= variable_list_attribute_redefinition_list
  variable_list_attribute_redefinition

```

```

variable_list_attribute_redefinition:
= members_redefinition
= help_redefinition
= optional_label_redefinition
= response_codes_reference_redefinition

```

```

waveform_redefinition:
= 'DELETE' 'WAVEFORM' Identifier ';'
= 'REDEFINE' 'WAVEFORM' Identifier '{' waveform_attribute_list '}'
= 'WAVEFORM' Identifier '{' waveform_attribute_redefinition_list '}'

```

```

waveform_attribute_redefinition_list:
= waveform_attribute_redefinition
= waveform_attribute_redefinition_list waveform_attribute_redefinition

```

```

waveform_attribute_redefinition:
= waveform_type_redefinition
= emphasis_redefinition
= waveform_exit_actions_redefinition
= handling_redefinition
= help_redefinition
= waveform_init_actions_redefinition
= waveform_key_points_redefinition
= optional_label_redefinition
= line_color_redefinition
= line_type_redefinition
= waveform_refresh_actions_redefinition
= validity_redefinition
= waveform_y_axis_redefinition

```

```

waveform_type_redefinition:
= 'REDEFINE' waveform_type
= 'TYPE' waveform_type_option_redefinition

```

```

waveform_type_option_redefinition:
= 'YT' '{' waveform_yt_type_option_redefinition_list '}'
= 'XY' '{' waveform_xy_type_option_redefinition_list '}'
= 'HORIZONTAL' '{' waveform_y_values_redefinition '}'
= 'VERTICAL' '{' waveform_x_values_redefinition '}'

```

```

waveform_yt_type_option_redefinition_list:
= waveform_yt_type_option_redefinition
= waveform_yt_type_option_redefinition_list
  waveform_yt_type_option_redefinition

```

```

waveform_yt_type_option_redefinition:
= yt_x_initial_redefinition
= yt_x_increment_redefinition
= waveform_y_values_redefinition
= number_of_points_redefinition

```

```

yt_x_initial_redefinition:
= 'REDEFINE' yt_x_initial

```

```

yt_x_increment_redefinition:
= 'REDEFINE' yt_x_increment

```

```

waveform_y_values_redefinition:
= 'REDEFINE' waveform_y_values

```

```

= 'ADD' waveform_y_values

waveform_x_values_redefinition:
= 'REDEFINE' waveform_x_values
= 'ADD' waveform_x_values

number_of_points_redefinition:
= 'DELETE' 'NUMBER_OF_POINTS' ';'
= 'REDEFINE' number_of_points

waveform_xy_type_option_redefinition_list:
= waveform_xy_type_option_redefinition
= waveform_xy_type_option_redefinition_list
  waveform_xy_type_option_redefinition

waveform_xy_type_option_redefinition:
= waveform_x_values_redefinition
= waveform_y_values_redefinition
= number_of_points_redefinition

waveform_exit_actions_redefinition:
= 'DELETE' 'EXIT_ACTIONS' ';'
= 'REDEFINE' waveform_exit_actions

waveform_init_actions_redefinition:
= 'DELETE' 'INIT_ACTIONS' ';'
= 'REDEFINE' waveform_init_actions

waveform_key_points_redefinition:
= 'DELETE' 'KEY_POINTS' ';'
= 'REDEFINE' 'KEY_POINTS' '{' key_points_options_redefinition_list '}'

key_points_options_redefinition_list:
= key_points_options_redefinition
= key_points_options_redefinition_list key_points_options_redefinition

key_points_options_redefinition:
= waveform_x_values_redefinition
= waveform_y_values_redefinition

waveform_refresh_actions_redefinition:
= 'DELETE' 'REFRESH_ACTIONS' ';'
= 'REDEFINE' waveform_refresh_actions

waveform_y_axis_redefinition:
= 'DELETE' 'Y_AXIS' ';'
= 'REDEFINE' waveform_y_axis

```

A.6.40 Références

```

reference:
= reference_without_call
= reference_without_call '(' argument_list ')'

reference_without_call:
= Identifier
= reference_without_call '[' expr ']'
= reference_without_call '.' Identifiant
= reference_without_call '.' selector
= reference_without_call '->' Identifier
= 'BLOCK' '.' Identifiant
= 'LOCAL_PARAM' '.' Identifiant
= 'PARAM' '.' Identifiant
= 'PARAM_LIST' '.' Identifiant

```

```

= block_a_reference '[' expr ']' '.' 'PARAM' '.' Identifiant
= block_a_reference '[' expr ']' '.' 'LOCAL_PARAM' '.' Identifiant
= block_a_reference '[' expr ']' '.' 'BLOCK' '.' Identifiant
= block_a_reference '[' expr ']' '.' 'CHART' '.' Identifiant
= block_a_reference '[' expr ']' '.' 'LIST' '.' Identifiant
= block_a_reference '[' expr ']' '.' 'GRAPH' '.' Identifiant
= block_a_reference '[' expr ']' '.' 'GRID' '.' Identifiant
= block_a_reference '[' expr ']' '.' 'MENU' '.' Identifiant
= block_a_reference '[' expr ']' '.' 'METHOD' '.' Identifiant

```

selector:

```

= 'CAPACITY'
= 'CONSTANT_UNIT'
= 'COUNT'
= 'DEFAULT_VALUE'
= 'FIRST'
= 'HELP'
= 'INITIAL_VALUE'
= 'LABEL'
= 'LAST'
= 'MAX_VALUE'
= 'MAX_VALUE1'
= 'MAX_VALUE2'
= 'MAX_VALUE3'
= 'MAX_VALUE4'
= 'MAX_VALUE5'
= 'MAX_VALUE6'
= 'MAX_VALUE7'
= 'MAX_VALUE8'
= 'MAX_VALUE9'
= 'MIN_VALUE'
= 'MIN_VALUE1'
= 'MIN_VALUE2'
= 'MIN_VALUE3'
= 'MIN_VALUE4'
= 'MIN_VALUE5'
= 'MIN_VALUE6'
= 'MIN_VALUE7'
= 'MIN_VALUE8'
= 'MIN_VALUE9'
= 'SCALING'
= 'SCALING_FACTOR'
= 'VARIABLE_STATUS'
= 'VIEW_MAX'
= 'VIEW_MIN'
= 'X_AXIS'
= 'Y_AXIS'

```

```

dictionary_reference:
= '[' Identifier ']'

```

```

array_reference:
= reference_without_call

```

```

axis_reference:
= reference_without_call

```

```

block_a_reference:
= reference_without_call

```

```

block_b_reference:
= reference_without_call

```

```
chart_reference:
  = reference_without_call

collection_reference:
  = reference_without_call

connection_reference:
  = reference_without_call

edit_display_reference:
  = reference_without_call

file_reference:
  = reference_without_call

graph_reference:
  = reference_without_call

grid_reference:
  = reference_without_call

image_reference:
  = reference_without_call

list_reference:
  = reference_without_call

item_array_reference:
  = reference_without_call

menu_reference:
  = reference_without_call

method_reference:
  = reference

record_reference:
  = reference_without_call

refresh_reference:
  = reference_without_call

response_code_reference:
  = reference_without_call

source_reference:
  = reference_without_call

unit_reference:
  = reference_without_call

variable_reference:
  = reference_without_call

wao_reference:
  = reference_without_call

waveform_reference:
  = reference_without_call
```

Annexe B (normative)

Bibliothèque de Builtin EDDL

B.1 Généralités

La présente annexe décrit les sous-programmes Builtin, qui peuvent être utilisés dans une EDD. Les Builtin prennent en charge les interactions avec l'utilisateur ou le dispositif. Des interactions types avec l'utilisateur sont, par exemple, des entrées par l'utilisateur de valeurs de VARIABLE ou d'accusés de réception. Des interactions types avec le dispositif sont, par exemple, l'envoi des commandes au dispositif ou des codes de réponse d'interprétation et des octets de statut. Cette annexe contient la bibliothèque de Builtin.

Outre les Builtin spécifiés dans ce document, des Builtin spécifiques à l'utilisateur peuvent être utilisés. Ces Builtin doivent être intégrés dans l'application EDD.

NOTE Les Builtin sont répertoriés dans l'ordre alphabétique.

B.2 Conventions pour les descriptions de Builtin

Les conventions suivantes s'appliquent pour les définitions de Builtin.

- Le Tableau B.1 définit le format pour les tableaux d'élément lexical de Builtin.
- Le Tableau B.2 définit le contenu des colonnes du Tableau B.1.

Tableau B.1 – Format pour les tableaux d'élément lexical de Builtin

Nom du paramètre	Type	Direction	Usage	Description

Tableau B.2 – Contenu du tableau d'élément lexical

Colonne	Texte	Signification
Nom du paramètre	<name>	Nom du paramètre formel.
	"<return>"	Valeur de retour du Builtin.
Type	data type	Type de données abstrait (basique ou dérivé) du paramètre formel.
Direction	I	Paramètre d'entrée.
	O	Paramètre de sortie.
	I/O	Paramètre d'entrée et de sortie.
	—	Aucune valeur n'est retournée (VOID).
Usage	m	Le paramètre formel est obligatoire.
	o	Le paramètre formel est facultatif.
	c	Le paramètre formel est conditionnel.
	—	Aucune valeur n'est retournée (VOID).
Description	<text>	Texte descriptif pour le paramètre.
	—	Aucune valeur n'est retournée (VOID).

B.3 Builtin _ERROR

Objet

Le Builtin _ERROR place une information textuelle dans un journal. Cette information est placée dans la catégorie d'erreur. Le Builtin _TRACE et le Builtin _WARNING écrivent des informations dans ce même journal.

Toute référence de variable contenue au format chaîne est développée en valeur réelle. Des variables locales de méthode et des variables DD peuvent être référencées dans le paramètre de message. Pour plus d'informations, consulter le Builtin PUT_MESSAGE.

Structure lexicale

`_WARNING(text, global_var_ids)`

Les éléments lexicaux du Builtin _TRACE sont présentés dans le Tableau B.3.

Tableau B.3 – Builtin _ERROR

Nom du paramètre	Type	Direction	Usage	Description
text	DD_STRING	I	m	Information textuelle à placer dans le journal.
global_var_ids	int[]	I	m	Matrice qui contient des identifiants uniques d'instance de VARIABLE.
<return>	VOID	—	—	—

B.4 Builtin _TRACE

Objet

Le Builtin _TRACE place une information textuelle dans un journal. Cette information est placée dans la catégorie trace. Le Builtin _ERROR et le Builtin _WARNING écrivent des informations dans ce même journal.

Toute référence de variable contenue au format chaîne est développée en valeur réelle. Des variables locales de méthode et des variables DD peuvent être référencées dans le paramètre de message. Pour plus d'informations, voir le Builtin PUT_MESSAGE.

Structure lexicale

`_TRACE(text, global_var_ids)`

Les éléments lexicaux du Builtin _TRACE sont présentés dans le Tableau B.4.

Tableau B.4 – Builtin _TRACE

Nom du paramètre	Type	Direction	Usage	Description
text	DD_STRING	I	m	Information textuelle à placer dans le journal.
global_var_ids	int[]	I	m	Matrice qui contient des identifiants uniques d'instance de VARIABLE.
<return>	VOID	—	—	—

B.5 Builtin _WARNING

Objet

Le Builtin _WARNING (avertissement) place une information textuelle dans un journal. Cette information est placée dans la catégorie warning (avertissement). Le Builtin _ERROR et le Builtin _TRACE écrivent des informations dans ce même journal.

Toute référence de variable contenue au format chaîne est développée en valeur réelle. Des variables locales de méthode et des variables DD peuvent être référencées dans le paramètre de message. Pour plus d'informations, voir le Builtin PUT_MESSAGE.

Structure lexicale

```
_WARNING(text, global_var_ids)
```

Les éléments lexicaux du Builtin _WARNING sont présentés dans le Tableau B.5.

Tableau B.5 – Builtin _WARNING

Nom du paramètre	Type	Direction	Usage	Description
text	DD_STRING	I	m	Information textuelle à placer dans le journal.
global_var_ids	int[]	I	m	Matrice qui contient des identifiants uniques d'instance de VARIABLE.
<return>	VOID	—	—	—

B.6 Builtin abort

Objet

Le Builtin abort (interrompre) affiche un message indiquant que la méthode a été interrompue et attend un accusé de réception de l'utilisateur. Une fois l'accusé de réception confirmé, le système exécute des méthodes d'interruption dans la liste de méthodes d'interruption et quitte la méthode.

Structure lexicale

```
abort(VOID)
```

Les éléments lexicaux du Builtin abort sont présentés dans le Tableau B.6.

Tableau B.6 – Builtin abort

Nom du paramètre	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.7 Builtin abort_on_all_comm_errors

Objet

Le Builtin abort_on_all_comm_errors spécifie la méthode à interrompre en cas de réception d'erreur de communication.

Lorsqu'une méthode est démarrée, elle n'est pas interrompue sur une erreur de communication à moins qu'abort_on_all_comm_errors soit appelé. Cela évite à une méthode de devoir gérer des codes d'erreur de communication en interrompant la méthode si elle reçoit une erreur de communication.

Après cet appel, tout appel de Builtin qui effectue une demande de communication qui retourne une erreur conduit à l'interruption de la méthode. Une erreur de communication zéro n'est pas considérée comme une erreur de communication.

Les codes d'erreur de communication sont définis par les consortiums. Leurs chaînes associées peuvent être définies dans un dictionnaire de texte.

Structure lexicale

`abort_on_all_comm_errors(VOID)`

L'élément lexical du Builtin `abort_on_all_comm_errors` est présenté dans le Tableau B.7.

Tableau B.7 – Builtin `abort_on_all_comm_errors`

Nom du paramètre	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.8 Builtin `ABORT_ON_ALL_COMM_STATUS`

Objet

Le Builtin `ABORT_ON_ALL_COMM_STATUS` définit tous les bits dans le masque communication status abort (interruption statut de communication). Cela conduit le système à interrompre la méthode actuelle si le dispositif retourne une valeur de statut de communication. Le statut de communication est défini comme étant le premier octet de données retourné dans une transaction, lorsque le bit 7 de cet octet est défini.

Les masques de relance et d'interruption sont réinitialisés à leurs valeurs par défaut au début de chaque méthode, de sorte que la nouvelle valeur du masque sera valide uniquement pendant la méthode actuelle. Voir le Builtin `ABORT_ON_RESPONSE_CODES` pour une liste des masques disponibles, et leurs valeurs par défaut.

Le masque affecté par cette fonction est utilisé par les Builtin `send`, `send_trans`, `send_command`, `send_command_trans`, `ext_send_command`, `ext_send_command_trans`, `get_more_status`, et `display`.

Structure lexicale

`ABORT_ON_ALL_COMM_STATUS(VOID)`

L'élément lexical du Builtin `ABORT_ON_ALL_COMM_STATUS` est présenté dans le Tableau B.8.

Tableau B.8 – Builtin `ABORT_ON_ALL_COMM_STATUS`

Nom du paramètre	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.9 Builtin `ABORT_ON_ALL_DEVICE_STATUS`

Objet

Le Builtin `ABORT_ON_ALL_DEVICE_STATUS` définit tous les bits dans le masque device status abort (interruption statut du dispositif). Cela conduit le système à interrompre la méthode actuelle si le dispositif retourne une valeur de statut du dispositif. Le statut du dispositif est défini comme étant le deuxième octet de données retourné dans une transaction.

Les masques de relance et d'interruption sont réinitialisés à leurs valeurs par défaut au début de chaque méthode, de sorte que la nouvelle valeur du masque sera valide uniquement pendant la méthode actuelle. Voir la fonction `send_command` pour l'implémentation des masques. Voir `ABORT_ON_RESPONSE_CODE` pour une liste des masques disponibles et leurs valeurs par défaut.

Le masque affecté par cette fonction est utilisé par les Builtin `send`, `send_trans`, `send_command`, `send_command_trans`, `ext_send_command`, `ext_send_command_trans`, `get_more_status`, et `display`.

Structure lexicale

`ABORT_ON_ALL_DEVICE_STATUS (VOID)`

L'élément lexical du Builtin `ABORT_ON_ALL_DEVICE_STATUS` est présenté dans le Tableau B.9.

Tableau B.9 – Builtin `ABORT_ON_ALL_DEVICE_STATUS`

Nom du paramètre	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.10 Builtin `ABORT_ON_ALL_RESPONSE_CODES`

Objet

Le Builtin `ABORT_ON_ALL_RESPONSE_CODES` définit tous les bits dans le masque réponse code abort (interruption code de réponse). Cela conduit le système à interrompre la méthode actuelle si le dispositif retourne une valeur code de réponse.

Le code de réponse est défini comme étant le premier octet de données retourné dans une transaction, lorsque le bit 7 de cet octet est 0.

Les masques de relance et d'interruption sont réinitialisés à leurs valeurs par défaut au début de chaque méthode, de sorte que la nouvelle valeur du masque sera valide uniquement pendant la méthode actuelle. Voir la fonction de commande d'envoi pour l'implémentation des masques. Voir `ABORT_ON_RESPONSE_CODE` pour une liste des masques disponibles et leurs valeurs par défaut.

Le masque affecté par cette fonction est utilisé par les Builtin `send`, `send_trans`, `send_command`, `send_command_trans`, `ext_send_command`, `ext_send_command_trans`, `get_more_status`, et `display`.

Structure lexicale

`ABORT_ON_ALL_RESPONSE_CODES (VOID)`

L'élément lexical du Builtin `ABORT_ON_ALL_RESPONSE_CODES` est présenté dans le Tableau B.10.

Tableau B.10 – Builtin `ABORT_ON_ALL_RESPONSE_CODES`

Nom du paramètre	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.11 Builtin abort_on_all_response_codes

Objet

Le Builtin `abort_on_all_response_codes` conduit à l'interruption de la méthode si celle-ci reçoit un code réponse différent de zéro, après l'appel d'un Builtin qui conduit à une demande à un dispositif. Ce Builtin évite à une méthode de devoir gérer des codes de réponse d'erreur. Lorsqu'une méthode est démarrée, elle n'est pas interrompue sur un code réponse jusqu'à ce que ce Builtin soit appelé. Un code réponse est une valeur représentant une condition d'erreur spécifique à une application. Une valeur code de réponse égale à zéro n'est pas considérée comme un code réponse d'erreur. Chaque code de réponse et type de code de réponse est défini dans EDDL.

Structure lexicale

`abort_on_all_response_codes (VOID)`

L'élément lexical du Builtin `abort_on_all_response_codes` est présenté dans le Tableau B.11.

Tableau B.11 – Builtin abort_on_all_response_codes

Nom du paramètre	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.12 Builtin ABORT_ON_COMM_ERROR

Objet

Le Builtin `ABORT_ON_COMM_ERROR` définit le masque no comm error (pas d'erreur de communication) de sorte que la méthode soit interrompue si une erreur de communication est détectée lors de l'envoi d'une commande.

Les masques de relance et d'interruption sont réinitialisés à leurs valeurs par défaut au début de chaque méthode, de sorte que la nouvelle valeur du masque sera valide uniquement pendant la méthode actuelle. Voir la fonction `send_command` pour l'implémentation des masques. Voir le Builtin `ABORT_ON_RESPONSE_CODES` pour une liste des masques disponibles et leurs valeurs par défaut.

Le masque affecté par cette fonction est utilisé par les Builtin `send`, `send_trans`, `send_command`, `send_command_trans`, `ext_send_command`, `ext_send_command_trans`, `get_more_status`, et `display`.

Structure lexicale

`ABORT_ON_COMM_ERROR (VOID)`

L'élément lexical du Builtin `ABORT_ON_COMM_ERROR` est présenté dans le Tableau B.12.

Tableau B.12 – Builtin ABORT_ON_COMM_ERROR

Nom du paramètre	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.13 Builtin abort_on_comm_error

Objet

Le Builtin abort_on_comm_error conduit à l'interruption de la méthode si celle-ci reçoit une erreur de communication spécifique.

Lorsqu'une méthode est démarrée, celle-ci n'est pas interrompue sur une erreur de communication spécifique à moins que le Builtin abort_on_comm_error ne soit appelé.

NOTE Une erreur de communication zéro n'est pas considérée comme une erreur de communication.

Ce Builtin permet à l'auteur de la méthode de sélectionner les erreurs de communication qu'il convient que la méthode gère et cela évite à une méthode de devoir gérer un code d'erreur de communication spécifique.

Les codes d'erreur de communication sont définis par les consortiums. Leurs chaînes associées peuvent être définies dans un dictionnaire de texte.

Structure lexicale

```
abort_on_comm_error(error)
```

Les éléments lexicaux du Builtin abort_on_comm_error sont présentés dans le Tableau B.13.

Tableau B.13 – Builtin abort_on_comm_error

Nom du paramètre	Type	Direction	Usage	Description
erreur	unsigned long	I	m	Code d'erreur de communication.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau B.346.

B.14 Builtin ABORT_ON_COMM_STATUS

Objet

Le Builtin ABORT_ON_COMM_STATUS définit le(s) bit(s) correct(s) dans le masque d'interruption comm_status (statut de communication) de sorte que la comm_status_value spécifiée conduise à l'interruption de la méthode. Le statut de communication est défini comme étant le premier octet de données retourné dans une transaction, lorsque le bit 7 de cet octet est positionné à 1.

Les masques de relance et d'interruption sont réinitialisés à leurs valeurs par défaut au début de chaque méthode, de sorte que la nouvelle valeur du masque sera valide uniquement pendant la méthode actuelle. Voir la fonction de commande d'envoi pour l'implémentation des masques. Voir ABORT_ON_RESPONSE_CODE pour une liste des masques disponibles et leurs valeurs par défaut.

Le masque affecté par cette fonction est utilisé par les Builtin send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, et display.

Structure lexicale

```
ABORT_ON_COMM_STATUS(comm_status)
```

Les éléments lexicaux du Builtin ABORT_ON_COMM_STATUS sont présentés dans le Tableau B.14.

Tableau B.14 – Builtin ABORT_ON_COMM_STATUS

Nom du paramètre	Type	Direction	Usage	Description
comm_status	unsigned char	I	m	Masque binaire pour le statut de communication.
<return>	VOID	—	—	—

B.15 Builtin ABORT_ON_DEVICE_STATUS**Objet**

Le Builtin ABORT_ON_DEVICE_STATUS définit le(s) bit(s) correct(s) dans le masque device status abort (interruption statut du dispositif) de sorte que la valeur de statut du dispositif spécifiée conduise à l'interruption de la méthode. Le statut du dispositif est défini comme étant le deuxième octet de données retourné dans une transaction.

Les masques de relance et d'interruption sont réinitialisés à leurs valeurs par défaut au début de chaque méthode, de sorte que la nouvelle valeur du masque sera valide uniquement pendant la méthode actuelle. Voir la fonction de commande d'envoi pour l'implémentation des masques. Voir ABORT_ON_RESPONSE_CODE pour une liste des masques disponibles et leurs valeurs par défaut.

Le masque affecté par cette fonction est utilisé par les Builtin send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, et display.

Structure lexicale

ABORT_ON_DEVICE_STATUS(device_status)

Les éléments lexicaux du Builtin ABORT_ON_DEVICE_STATUS sont présentés dans le Tableau B.15.

Tableau B.15 – Builtin ABORT_ON_DEVICE_STATUS

Nom du paramètre	Type	Direction	Usage	Description
device_status	unsigned char	I	m	Masque binaire pour le statut du dispositif.
<return>	VOID	—	—	—

B.16 Builtin ABORT_ON_NO_DEVICE**Objet**

Le Builtin ABORT_ON_NO_DEVICE définit le masque no devices de sorte que la méthode soit interrompue si aucun dispositif n'est détecté lors de l'envoi d'une commande.

Les masques de relance et d'interruption sont réinitialisés à leurs valeurs par défaut au début de chaque méthode, de sorte que la nouvelle valeur du masque sera valide uniquement pendant la méthode actuelle. Voir la fonction de commande d'envoi pour l'implémentation des masques. Voir ABORT_ON_RESPONSE_CODE pour une liste des masques disponibles et leurs valeurs par défaut.

Le masque affecté par cette fonction est utilisé par les Builtin send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, et display.

Structure lexicale

ABORT_ON_NO_DEVICE (VOID)

L'élément lexical du Builtin ABORT_ON_NO_DEVICE est présenté dans le Tableau B.16.

Tableau B.16 – Builtin ABORT_ON_NO_DEVICE

Nom du paramètre	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.17 Builtin ABORT_ON_RESPONSE_CODE

Objet

Le Builtin ABORT_ON_RESPONSE_CODE définit le(s) bit(s) correct(s) dans le masque d'interruption de code de réponse de sorte que la valeur de code de réponse spécifiée conduise à l'interruption de la méthode. Le code de réponse est défini comme étant le premier octet de données retourné dans une transaction, lorsque le bit 7 de cet octet est 0.

Les masques de relance et d'interruption sont réinitialisés à leurs valeurs par défaut au début de chaque méthode, de sorte que la nouvelle valeur du masque sera valide uniquement pendant la méthode actuelle.

Une liste de l'ensemble des masques disponibles est présentée ci-dessous. Les valeurs par défaut des masques sont les suivantes:

- response_code_abort_mask = ACCESS_RESTRICTED | CMD_NOT_IMPLEMENTED
- response_code_retry_mask = NO BITS SET
- cmd_48_response_code_abort_mask = ACCESS_RESTRICTED | CMD_NOT_IMPLEMENTED
- cmd_48_response_code_retry_mask = NO BITS SET
- device_status_abort_mask = DEVICE_MALFUNCTION
- device_status_retry_mask = NO BITS SET
- cmd_48_device_status_abort_mask = DEVICE_MALFUNCTION
- cmd_48_device_status_retry_mask = NO BITS SET
- comm_status_abort_mask = NO BITS SET
- comm_status_retry_mask = ALL BITS SET
- cmd_48_comm_status_abort_mask = NO BITS SET
- cmd_48_comm_status_retry_mask = ALL BITS SET
- no_device_abort_mask = ALL BITS SET
- comm_error_abort_mask = ALL BITS SET
- comm_error_retry_mask = NO BITS SET
- cmd_48_no_device_abort_mask = ALL BITS SET
- cmd_48_comm_error_abort_mask = ALL BITS SET
- cmd_48_comm_error_retry_mask = NO BITS SET
- cmd_48_data_abort_mask = NO BITS SET
- cmd_48_data_retry_mask = NO BITS SET

Le masque affecté par cette fonction est utilisé par les Builtin `send`, `send_trans`, `send_command`, `send_command_trans`, `ext_send_command`, `ext_send_command_trans`, `get_more_status`, et `display`.

Structure lexicale

`ABORT_ON_RESPONSE_CODE(response_code)`

Les éléments lexicaux du Builtin `ABORT_ON_RESPONSE_CODE` sont présentés dans le Tableau B.17.

Tableau B.17 – Builtin `ABORT_ON_RESPONSE_CODE`

Nom du paramètre	Type	Direction	Usage	Description
<code>response_code</code>	<code>int</code>	<code>l</code>	<code>m</code>	Masque binaire pour le code de réponse.
<code><return></code>	<code>VOID</code>	—	—	—

B.18 Builtin `abort_on_response_code`

Objet

Le Builtin `abort_on_response_code` conduit à l'interruption de la méthode après réception du code de réponse spécifié.

Le Builtin conduit à l'interruption de la méthode si celle-ci reçoit un code de réponse spécifique. Ce Builtin évite à une méthode de devoir gérer un code de réponse spécifique.

Lorsqu'une méthode est démarrée, celle-ci n'est pas interrompue sur un code de réponse spécifique à moins que ce Builtin ne soit appelé. Un code de réponse est une valeur représentant une condition d'erreur spécifique à une application.

NOTE Une valeur de code de réponse de zéro n'est pas considérée comme un code de réponse d'erreur. Chaque type de code de réponse est défini dans l'EDD.

Structure lexicale

`abort_on_response_code(code)`

Les éléments lexicaux du Builtin `abort_on_response_code` sont présentés dans le Tableau B.18.

Tableau B.18 – Builtin `abort_on_response_code`

Nom du paramètre	Type	Direction	Usage	Description
<code>code</code>	<code>unsigned long</code>	<code>l</code>	<code>m</code>	Masque binaire pour le code de réponse.
<code><return></code>	<code>long</code>	<code>O</code>	<code>m</code>	Un des codes de retour spécifiés dans le Tableau B.346.

B.19 Builtin `abs`

Objet

Le Builtin `abs` retourne la valeur absolue d'une valeur à virgule flottante.

Structure lexicale

`abs(x)`

Les éléments lexicaux du Builtin `abs` sont présentés dans le Tableau B.19.

Tableau B.19 – Builtin abs

Nom du paramètre	Type	Direction	Usage	Description
x	double	I	m	Valeur à virgule flottante.
<return>	double	O	m	Valeur absolue.

B.20 Builtin ACKNOWLEDGE**Objet**

Le Builtin ACKNOWLEDGE (accuser réception) affiche l'invite et attend la pression de la touche Entrée. Contrairement au Builtin acknowledge, l'invite ne peut PAS contenir de VARIABLE du dispositif incorporées.

Structure lexicale

ACKNOWLEDGE(prompt)

Les éléments lexicaux du Builtin ACKNOWLEDGE sont présentés dans le Tableau B.20.

Tableau B.20 – Builtin ACKNOWLEDGE

Nom du paramètre	Type	Direction	Usage	Description
prompt	char[]	I	m	Message à afficher.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau B.347

B.21 Builtin acknowledge**Objet**

Le Builtin acknowledge affiche l'invite et attend l'accusé de réception de l'utilisateur. L'invite peut contenir des valeurs de variables locales et/ou du dispositif.

Structure lexicale

acknowledge(prompt, global_var_ids)

Les éléments lexicaux du Builtin acknowledge sont présentés dans le Tableau B.21.

Tableau B.21 – Builtin acknowledge

Nom du paramètre	Type	Direction	Usage	Description
prompt	char[]	I	m	Message à afficher.
global_var_ids	matrice d'entiers	I	m	Matrice contenant des identifiants uniques d'instances de VARIABLE.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau B.347.

B.22 Builtin acknowledge**Objet**

Le Builtin acos retourne l'arc cosinus d'une valeur à virgule flottante.

Structure lexicale

`acos (x)`

Les éléments lexicaux du Builtin `acos` sont présentés dans le Tableau B.22.

Tableau B.22 – Builtin `acos`

Nom du paramètre	Type	Direction	Usage	Description
<code>x</code>	double	I	m	Valeur à virgule flottante.
<code><return></code>	double	O	m	Arc cosinus.

B.23 Builtin `add_abort_method` (version A)

Objet

Le Builtin `add_abort_method` ajoute une méthode à la liste de méthodes d'interruption, qui est la liste des méthodes à exécuter si la méthode actuelle est interrompue. La liste de méthodes d'interruption peut contenir jusqu'à 20 méthodes. Les méthodes sont exécutées dans l'ordre dans lequel elles sont ajoutées à la liste, et la même méthode peut être ajoutée à la liste plus d'une fois. La liste est effacée après chaque exécution de la méthode `non_abort`.

`add_abort_method` est exécuté uniquement lorsque la méthode est interrompue, et non lorsque la méthode se termine dans des conditions opératoires normales. Des méthodes peuvent être interrompues en raison d'une condition de masque d'interruption lors de l'envoi d'une commande, ou lorsque le Builtin `abort` est appelé.

Structure lexicale

`add_abort_method(abort_method_id)`

Les éléments lexicaux du Builtin `add_abort_method` sont présentés dans le Tableau B.23.

Tableau B.23 – Builtin `add_abort_method`

Nom du paramètre	Type	Direction	Usage	Description
<code>abort_method_id</code>	reference	I	m	Nom d'une instance de <code>METHOD</code> .
<code><return></code>	int	O	m	Un des codes de retour spécifiés dans le Tableau B.347.

B.24 Builtin `add_abort_method` (version B)

Objet

Le Builtin `add_abort_method` ajoute une méthode d'interruption à la liste de méthodes d'interruption de la méthode appelante. La liste de méthodes d'interruption est la liste des méthodes qui sont exécutées si la méthode est interrompue. La liste de méthodes d'interruption s'applique uniquement à l'exécution en cours de la méthode.

Les méthodes sont interrompues lorsque des codes de réponse ou des erreurs de communication surviennent qui ont été configurés pour déclencher une interruption via d'autres appels de Builtin, par exemple, `abort_on_all_comm_error` et `abort_on_all_response_codes`. Des méthodes peuvent également être explicitement interrompues par un appel direct du Builtin `method_abort`, ou lorsqu'une erreur se produit. La terminaison normale d'une méthode n'est pas considérée comme une interruption.

La liste de méthodes d'interruption est initialement vide lorsqu'une méthode commence son exécution. Les méthodes d'interruption sont ajoutées à la fin de la liste et sont supprimées et

exécutées depuis la tête de la liste dans l'ordre dans lequel elles apparaissent, c'est-à-dire, selon la règle premier entré/premier sorti. Une méthode d'interruption peut apparaître plus d'une fois dans la liste.

Les méthodes d'interruption peuvent être partagées entre des méthodes, ce qui signifie que deux méthodes peuvent avoir la même méthode d'interruption dans leurs listes de méthodes d'interruption respectives.

Le paramètre `method_id` est spécifié par l'auteur de la méthode en utilisant le Builtin `ITEM_ID` et le nom de la méthode. Les Builtin de méthodes d'interruption ne peuvent pas appeler `add_abort_method`.

Structure lexicale

`add_abort_method(method_id)`

Les éléments lexicaux du Builtin `add_abort_method` sont présentés dans le Tableau B.24.

Tableau B.24 – Builtin `add_abort_method`

Nom du paramètre	Type	Direction	Usage	Description
<code>method_id</code>	unsigned long	I	m	ID de la méthode.
<code><return></code>	long	O	m	Un des codes de retour spécifiés dans le Tableau B.346

B.25 Builtin `AddTime`

Objet

Le Builtin `AddTime` ajoute un nombre de secondes à une valeur `time_t`.

Structure lexicale

`AddTime(t0, seconds)`

Les éléments lexicaux du Builtin `AddTime` sont présentés dans le Tableau B.25.

Tableau B.25 – Builtin `AddTime`

Nom du paramètre	Type	Direction	Usage	Description
<code>t0</code>	<code>time_t</code>	I	m	Valeur <code>time_t</code> initiale.
<code>seconds</code>	double	I	m	Nombre de secondes à ajouter.
<code><return></code>	<code>time_t</code>	O	m	Valeur <code>time_t</code> spécifiée.

B.26 Builtin `asin`

Objet

Le Builtin `asin` retourne l'arc sinus d'une valeur à virgule flottante.

Structure lexicale

`asin(x)`

Les éléments lexicaux du Builtin `asin` sont présentés dans le Tableau B.26.

Tableau B.26 – Builtin asin

Nom du paramètre	Type	Direction	Usage	Description
x	double	I	m	Valeur à virgule flottante.
<return>	double	O	m	Arc sinus.

B.27 Builtin assign

Objet

Le Builtin assign définit la valeur d'une VARIABLE à la valeur d'une autre VARIABLE. Il définit la valeur de la VARIABLE de destination à une valeur égale à la valeur de la VARIABLE source.

NOTE Le Builtin est un moyen raccourci d'effectuer un ABN_GET_TYPE VALUE suivi par une commande put_type_value.

Les paramètres dst_id, src_id, dst_member_id, et src_member_id sont spécifiés en utilisant les Builtin ITEM_ID et MEMBER ID et la référence de la variable. La VARIABLE source et la VARIABLE de destination doivent être du même type.

Structure lexicale

assign(dst_id, dst_member_id, src_id, src_member_id)

Les éléments lexicaux du Builtin assign sont présentes dans le Tableau B.27.

Tableau B.27 – Builtin assign

Nom du paramètre	Type	Direction	Usage	Description
dst_id	unsigned long	I	m	ID d'item de la variable de destination.
dst_member_id	unsigned long	I	m	ID de membre de la variable de destination.
src_id	unsigned long	I	m	ID d'item de la variable source.
src_member_id	unsigned long	I	m	ID de membre de la variable source.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau B.346.

B.28 Builtin assign_double

Objet

Le Builtin assign_double assigne une valeur à une VARIABLE.

Structure lexicale

assign_double(device_var, new_value)

Les éléments lexicaux du Builtin assign_double sont présentés dans le Tableau B.28.

Tableau B.28 – Builtin assign_double

Nom du paramètre	Type	Direction	Usage	Description
device_var	reference	I	m	Nom de la variable.
new_value	double	I	m	Nouvelle valeur de la variable référencée.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau B.347.

B.29 Builtin assign_float**Objet**

Le Builtin assign_float assigne une valeur à une VARIABLE.

Structure lexicale

```
assign_float(device_var, new_value)
```

Les éléments lexicaux du Builtin assign_float sont présentes dans le Tableau B.29.

Tableau B.29 – Builtin assign_float

Nom du paramètre	Type	Direction	Usage	Description
device_var	reference	I	m	Nom de la variable.
new_value	double	I	m	Nouvelle valeur de la variable référencée.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau B.347

B.30 Builtin assign_int**Objet**

Le Builtin assign_int assigne une valeur à une VARIABLE.

Structure lexicale

```
assign_int(device_var, new_value)
```

Les éléments lexicaux du Builtin assign_int sont présentés dans le Tableau B.30.

Tableau B.30 – Builtin assign_int

Nom du paramètre	Type	Direction	Usage	Description
device_var	reference	I	m	Nom de la variable.
new_value	int	I	m	Nouvelle valeur de la variable référencée.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau B.347

B.31 Builtin assign_var**Objet**

Le Builtin assign_var assigne une valeur de la VARIABLE source à la VARIABLE de destination. Les deux variables doivent être du même type.

Structure lexicale

```
assign_var(destination_var, source_var)
```

Les éléments lexicaux du Builtin assign_var sont présentés dans le Tableau B.31.

Tableau B.31 – Builtin assign_var

Nom du paramètre	Type	Direction	Usage	Description
destination_var	reference	I	m	Nom de la variable de destination.
source_var	reference	I	m	Nom de la variable source.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau B.347.

B.32 Builtin assign2

Objet

Le Builtin assign2 définit la valeur d'une VARIABLE à la valeur d'une autre VARIABLE. Il définit la valeur de la VARIABLE de destination à une valeur égale à la valeur de la VARIABLE source.

Les paramètres dst_id, src_id, dst_member_id, et src_member_id sont spécifiés en utilisant les Builtin ITEM_ID et MEMBER ID et la référence de la VARIABLE. La VARIABLE source et la VARIABLE de destination doivent être du même type.

Structure lexicale

```
assign2(dst_blk_id, dst_blk_num, dst_id, dst_member_id, src_blk_id, src_blk_num, src_id, src_member_id)
```

Les éléments lexicaux du Builtin assign2 sont présentés dans le Tableau B.32.

Tableau B.32 – Builtin assign2

Nom du paramètre	Type	Direction	Usage	Description
dst_blk_id	unsigned long	I	m	ID d'item du bloc de destination.
dst_blk_num	unsigned long	I	m	Numéro d'occurrence du bloc de destination.
dst_id	unsigned long	I	m	ID d'item de la variable de destination.
dst_member_id	unsigned long	I	m	ID de membre de la variable de destination.
src_blk_id	unsigned long	I	m	ID d'item du bloc source.
src_blk_num	unsigned long	I	m	Numéro d'occurrence du bloc source.
src_id	unsigned long	I	m	ID d'item de la variable source.
src_member_id	unsigned long	I	m	ID de membre de la variable source.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau B.346.

B.33 Builtin atan

Objet

Le Builtin atan retourne l'arc tangente d'une valeur à virgule flottante.

Structure lexicale

`atan(x)`

Les éléments lexicaux du Builtin atan sont présentés dans le Tableau B.33.

Tableau B.33 – Builtin atan

Nom du paramètre	Type	Direction	Usage	Description
x	double	I	m	Valeur à virgule flottante.
<return>	double	O	m	Arc tangente.

B.34 Builtin ATOF

Objet

Le Builtin ATOF convertit une chaîne en valeur à virgule flottante.

Structure lexicale

`ATOF(string)`

Les éléments lexicaux du Builtin ATOF sont présentés dans le Tableau B.34.

Tableau B.34 – Builtin ATOF

Nom du paramètre	Type	Direction	Usage	Description
string	DD_STRING	I	m	Chaîne à convertir.
<return>	double	O	m	Valeur à virgule flottante.

B.35 Builtin atof

Objet

Le Builtin atof convertit un texte en valeur à virgule flottante.

Structure lexicale

`atof(str)`

Les éléments lexicaux du Builtin atof sont présentés dans le Tableau B.35.

Tableau B.35 – Builtin atof

Nom du paramètre	Type	Direction	Usage	Description
str	char[]	I	m	Texte à convertir.
<return>	float	O	m	Valeur à virgule flottante.

B.36 Builtin ATOI

Objet

Le Builtin ATOI convertit une chaîne en valeur entière.

Structure lexicale

ATOI(str)

Les éléments lexicaux du Builtin ATOI sont présentés dans le Tableau B.36.

Tableau B.36 – Builtin ATOI

Nom du paramètre	Type	Direction	Usage	Description
str	DD_STRING	I	m	Chaîne à convertir.
<return>	int	O	m	Valeur entière.

B.37 Builtin atoi

Objet

Le Builtin convertit un texte en valeur entière.

Structure lexicale

atoi(str)

Les éléments lexicaux du Builtin atoi sont présentés dans le Tableau B.37.

Tableau B.37 – Builtin atoi

Nom du paramètre	Type	Direction	Usage	Description
str	char[]	I	m	Texte à convertir.
<return>	int	O	m	Valeur entière.

B.38 Builtin BUILD_MESSAGE

Objet

Le Builtin BUILD_MESSAGE retourne une chaîne basée sur une spécification de format. Toute référence de variable contenue au format chaîne est développée en valeur réelle. Des variables locales de méthode et des variables DD peuvent être référencées dans le paramètre de message. Pour plus d'informations, voir le Builtin PUT_MESSAGE.

Structure lexicale

BUILD_MESSAGE(string)

Les éléments lexicaux du Builtin BUILD_MESSAGE sont présentés dans le Tableau B.38.

Tableau B.38 – Builtin BUILD_MESSAGE

Nom du paramètre	Type	Direction	Usage	Description
string	DD_STRING	I	m	Spécification de format.
<return>	DD_STRING	O	m	Chaîne formatée.

B.39 Builtin ByteToDouble

Objet

Le Builtin ByteToDouble retourne une valeur double précision à partir d'un vecteur de huit octets.

Structure lexicale

ByteToDouble(in1, in2, in3, in4, in5, in6, in7, in8)

Les éléments lexicaux du Builtin ByteToDouble sont présentés dans le Tableau B.39.

Tableau B.39 – Builtin ByteToDouble

Nom du paramètre	Type	Direction	Usage	Description
in1	unsigned char	l	m	Premier élément de la valeur double précision (octet le plus significatif).
in2	unsigned char	l	m	Deuxième élément de la valeur double précision.
in3	unsigned char	l	m	Troisième élément de la valeur double précision.
in4	unsigned char	l	m	Quatrième élément de la valeur double précision.
in5	unsigned char	l	m	Cinquième élément de la valeur double précision.
in6	unsigned char	l	m	Sixième élément de la valeur double précision.
in7	unsigned char	l	m	Septième élément de la valeur double précision.
in8	unsigned char	l	m	Huitième élément de la valeur double précision (octet le moins significatif).
<return>	double	O	m	Un des codes de retour spécifiés dans le Tableau B.347.

B.40 Builtin ByteToFloat

Objet

Le Builtin ByteToFloat retourne une valeur à virgule flottante à partir d'un vecteur de quatre octets.

Structure lexicale

ByteToFloat(in1, in2, in3, in4)

Les éléments lexicaux du Builtin ByteToFloat sont présentés dans le Tableau B.40.

Tableau B.40 – Builtin ByteToFloat

Nom du paramètre	Type	Direction	Usage	Description
in1	unsigned char	l	m	Premier élément de la valeur à virgule flottante (octet le plus significatif).
in2	unsigned char	l	m	Deuxième élément de la valeur à virgule flottante.
in3	unsigned char	l	m	Troisième élément de la valeur à virgule flottante.
in4	unsigned char	l	m	Quatrième élément de la valeur à virgule flottante (octet le moins significatif).

Nom du paramètre	Type	Direction	Usage	Description
<return>	float	O	m	Un des codes de retour spécifiés dans le Tableau B.347.

B.41 Builtin ByteToLong

Objet

Le Builtin ByteToLong retourne une valeur longue à partir d'un vecteur de quatre octets.

Structure lexicale

ByteToLong(in1, in2, in3, in4)

Les éléments lexicaux du Builtin ByteToLong sont présentés dans le Tableau B.41.

Tableau B.41 – Builtin ByteToLong

Nom du paramètre	Type	Direction	Usage	Description
in1	unsigned char	I	m	Premier élément de la valeur longue (octet le plus significatif).
in2	unsigned char	I	m	Deuxième élément de la valeur longue.
in3	unsigned char	I	m	Troisième élément de la valeur longue.
in4	unsigned char	I	m	Quatrième élément de la valeur longue (octet le moins significatif).
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau B.347

B.42 Builtin ByteToShort

Objet

Le Builtin ByteToShort retourne une valeur courte à partir d'un vecteur de deux octets.

Structure lexicale

ByteToShort(in1, in2)

Les éléments lexicaux du Builtin ByteToShort sont présentés dans le Tableau B.42.

Tableau B.42 – Builtin ByteToShort

Nom du paramètre	Type	Direction	Usage	Description
in1	unsigned char	I	m	Premier élément de la valeur courte (octet le plus significatif).
in2	unsigned char	I	m	Deuxième élément de la valeur courte (octet le moins significatif).
<return>	short	O	m	Un des codes de retour spécifiés dans le Tableau B.347

B.43 Builtin cbrt

Objet

Le Builtin cbrt retourne la racine cubique d'une valeur à virgule flottante.

Structure lexicale`cbrt(x)`

Les éléments lexicaux du Builtin `cbrt` sont présentés dans le Tableau B.43.

Tableau B.43 – Builtin `cbrt`

Nom du paramètre	Type	Direction	Usage	Description
x	double	I	m	Valeur à virgule flottante.
<return>	double	O	m	Racine cubique.

B.44 Builtin `ceil`**Objet**

Le Builtin `ceil` retourne la plus petite valeur entière non inférieure à une valeur à virgule flottante spécifiée (valeur "plafond").

Structure lexicale`ceil(x)`

Les éléments lexicaux du Builtin `ceil` sont présentés dans le Tableau B.44.

Tableau B.44 – Builtin `ceil`

Nom du paramètre	Type	Direction	Usage	Description
x	double	I	m	Valeur à virgule flottante.
<return>	double	O	m	Valeur plafond.

B.45 Builtin `cos`**Objet**

Le Builtin `cos` retourne le cosinus d'une valeur à virgule flottante.

Structure lexicale`cos(x)`

Les éléments lexicaux du Builtin `cos` sont présentés dans le Tableau B.45.

Tableau B.45 – Builtin `cos`

Nom du paramètre	Type	Direction	Usage	Description
x	double	I	m	Valeur à virgule flottante.
<return>	double	O	m	Cosinus.

B.46 Builtin `cosh`**Objet**

Le Builtin `cosh` retourne le cosinus hyperbolique d'une valeur à virgule flottante.

Structure lexicale

`cosh(x)`

Les éléments lexicaux du Builtin `cosh` sont présentés dans le Tableau B.46.

Tableau B.46 – Builtin `cosh`

Nom du paramètre	Type	Direction	Usage	Description
<code>x</code>	double	I	m	Valeur à virgule flottante.
<return>	double	O	m	Cosinus hyperbolique.

B.47 Builtin `dassign`

Objet

Le Builtin `dassign` assigne une valeur à une VARIABLE.

Structure lexicale

`dassign(device_var, new_value)`

Les éléments lexicaux du Builtin `dassign` sont présentés dans le Tableau B.47.

Tableau B.47 – Builtin `dassign`

Nom du paramètre	Type	Direction	Usage	Description
<code>device_var</code>	reference	I	m	Nom de la variable.
<code>new_value</code>	double	I	m	Nouvelle valeur de la variable référencée.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau B.347.

B.48 Builtin `DATE_AND_TIME_VALUE_to_string`

Objet

Le Builtin `DATE_AND_TIME_VALUE_to_string` crée une représentation de chaîne d'une DATE et une TIME_VALUE (valeur de temps). La fonction ANSI C `strftime` définit la structure du format de chaîne.

Structure lexicale

`DATE_AND_TIME_VALUE_to_string(string, format, date, time_value)`

Les éléments lexicaux du Builtin `DATE_AND_TIME_VALUE_to_string` sont présentés dans le Tableau B.48.

Tableau B.48 – Builtin `DATE_AND_TIME_VALUE_to_string`

Nom du paramètre	Type	Direction	Usage	Description
<code>string</code>	DD_STRING	I	m	Chaîne dans laquelle les DATE et TIME_VALUE formatées sont copiées.
<code>format</code>	DD_STRING	I	m	Indique comment les DATE et TIME_VALUE doivent être formatées.
<code>date</code>	long	I	m	DATE à formater.
<code>time_value</code>	unsigned long	I	M	TIME_VALUE à formater.
<return>	int	O	m	Nombre de caractères réellement copiés ou -1 en cas

Nom du paramètre	Type	Direction	Usage	Description
				d'erreur.

B.49 Builtin Date_to_DayOfMonth

Objet

Le Builtin Date_to_DayOfMonth calcule le jour du mois à partir d'un nombre de jours donnés depuis le 1^{er} janvier 1900.

Structure lexicale

Date_to_DayOfMonth(date)

Les éléments lexicaux du Builtin Date_to_DayOfMonth sont présentés dans le Tableau B.49.

Tableau B.49 – Builtin Date_to_DayOfMonth

Nom du paramètre	Type	Direction	Usage	Description
date	long	I	m	Date. L'application EDD doit prendre en charge une correspondance implicite entre le format long et DATE, de sorte que ce Builtin accepte également des types DATE.
<return>	int	O	m	Jour du mois.

B.50 Builtin DATE_to_days

Objet

Le Builtin DATE_to_days calcule le nombre de jours entre deux dates.

Structure lexicale

DATE_to_days(date1, date0)

Les éléments lexicaux du Builtin DATE_to_days sont présentés dans le Tableau B.50.

Tableau B.50 – Builtin DATE_to_days

Nom du paramètre	Type	Direction	Usage	Description
date1	long	I	m	Date la plus récente.
date0	long	I	m	Date la moins récente.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau B.347.

B.51 Builtin Date_to_Month

Objet

Le Builtin Date_to_Month calcule le mois à partir d'un nombre donné de jours depuis le 1^{er} janvier 1900.

Structure lexicale

Date_to_Month(date)

Les éléments lexicaux du Builtin Date_to_Month sont présentés dans le Tableau B.51.

Tableau B.51 – Builtin Date_to_Month

Nom du paramètre	Type	Direction	Usage	Description
date	long	I	m	Nombre de jours. L'application EDD doit prendre en charge une correspondance implicite entre le format long et DATE, de sorte que ce Builtin accepte également des types DATE.
<return>	int	O	m	Mois.

B.52 Builtin DATE_to_string

Purpose

Le Builtin DATE_to_string crée une représentation de chaîne d'une DATE. La fonction strftime ANSI C définit la structure du format chaîne.

Structure lexicale

DATE_to_string(string, format, date)

Les éléments lexicaux du Builtin DATE_to_string sont présentés dans le Tableau B.52.

Tableau B.52 – Builtin DATE_to_string

Nom du paramètre	Type	Direction	Usage	Description
string	DD_STRING	I	m	Chaîne dans laquelle la DATE formatée est copiée.
format	DD_STRING	I	m	Indique comment la DATE doit être formatée.
date	long	I	m	DATE à formater.
<return>	int	O	m	Nombre de caractères effectivement copiés ou -1 en cas d'erreur.

B.53 Builtin Date_To_Time

Objet

Le Builtin Date_To_Time crée un time_t à partir d'une valeur de date. L'heure du jour est définie à 00:00:00.

Structure lexicale

Date_To_Time(date)

Les éléments lexicaux du Builtin Date_To_Time sont présentés dans le Tableau B.53.

Tableau B.53 – Builtin Date_To_Time

Nom du paramètre	Type	Direction	Usage	Description
date	long	I	m	Valeur de DATE.
<return>	time_t	O	m	Valeur de time_t.

B.54 Builtin Date_to_Year

Objet

Le Builtin Date_to_Year calcule l'année à partir d'un nombre donné de jours depuis le 1^{er} janvier 1900.

Structure lexicale

Date_to_Year(date)

Les éléments lexicaux du Builtin Date_to_Year sont présentés dans le Tableau B.54.

Tableau B.54 – Builtin Date_to_Year

Nom du paramètre	Type	Direction	Usage	Description
date	long	I	m	Nombre de jours. L'application EDD doit prendre en charge une correspondance implicite entre le format long et DATE, de sorte que ce Builtin accepte également des types DATE.
<return>	int	O	m	Mois depuis le 1 ^{er} janvier 1900.

B.55 Builtin days_to_DATE**Objet**

Le Builtin days_to_DATE calcule une date future en ajoutant un nombre de jours à une DATE.

Structure lexicale

days_to_DATE(days, date)

L'élément lexical du Builtin days_to_DATE est présenté dans le Tableau B.55.

Tableau B.55 – Builtin days_to_DATE

Nom du paramètre	Type	Direction	Usage	Description
days	long	I	m	Nombre de jours à ajouter à la DATE.
date	long	I	m	DATE initiale.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau B.347.

B.56 Builtin DELAY**Objet**

Le Builtin DELAY affiche un message d'invite et reste en pause pendant le nombre spécifié de secondes. L'invite peut contenir des valeurs de variable locale (voir Builtin put_message pour la structure lexicale). L'invite ne peut PAS contenir des valeurs de variable du dispositif. Le délai doit être un nombre positif.

NOTE 1 Il n'est pas garanti que les messages précédemment affichés restent à l'écran.

NOTE 2 Certaines applications EDD effacent l'affichage à chaque appel de cette fonction.

Structure lexicale

DELAY(delay_time, prompt)

Les éléments lexicaux du Builtin DELAY sont présentés dans le Tableau B.56.

Tableau B.56 – Builtin DELAY

Nom du paramètre	Type	Direction	Usage	Description
delay_time	unsigned integer	I	m	Délai en nombre de secondes.
prompt	char[]	I	m	Message à afficher.
<return>	VOID	—	—	—

B.57 Builtin delay

Objet

Le Builtin delay affiche un message d'invite, et reste en pause pendant le nombre spécifié de secondes. L'invite peut contenir des valeurs de variable locale et/ou du dispositif (voir put_message pour la structure lexicale). Le délai doit être un nombre positif.

NOTE 1 Il n'est pas garanti que les messages précédemment affichés restent à l'écran.

NOTE 2 Certaines applications EDD effacent l'affichage à chaque appel de cette fonction.

Structure lexicale

delay(delay_time, prompt, global_var_ids)

Les éléments lexicaux du Builtin delay sont présentés dans le Tableau B.57.

Tableau B.57 – Builtin delay

Nom du paramètre	Type	Direction	Usage	Description
delay_time	unsigned integer	I	m	Délai en nombre de secondes.
prompt	char[]	I	m	Message à afficher.
global_var_ids	array of int		m	Matrice contenant des identifiants uniques d'instances de VARIABLE.
<return>	VOID	—	—	—

B.58 Builtin DELAY_TIME

Objet

Le Builtin DELAY_TIME reste en pause pendant le nombre spécifié de secondes. Le délai doit être un nombre positif.

Structure lexicale

DELAY_TIME(delay_time)

Les éléments lexicaux du Builtin DELAY_TIME sont présentés dans le Tableau B.58.

Tableau B.58 – Builtin DELAY_TIME

Nom du paramètre	Type	Direction	Usage	Description
delay_time	unsigned integer	I	m	Délai en secondes.
<return>	VOID	—	—	—

B.59 Builtin delayfor

Objet

Le Builtin delayfor imprime une chaîne sur l'écran d'application et attend que la durée allouée expire. Les valeurs de VARIABLE peuvent être incorporées dans la chaîne de message avec des informations de formatage.

Structure lexicale

```
delayfor(duration, prompt, ids, indices, id_count)
```

Les éléments lexicaux du Builtin delayfor sont présentés dans le Tableau B.59.

Tableau B.59 – Builtin delayfor

Nom du paramètre	Type	Direction	Usage	Description
duration	long	I	m	Délai en secondes.
prompt	char[]	I	m	Message à afficher.
ids	array of unsigned long	I	m	ID d'élément d'une instance de VARIABLE utilisée dans le message d'invite.
indices	array of unsigned long	I	m	ID de membre ou indices de matrice des variables utilisées dans le message d'invite. Défini à zéro s'il n'y a pas d>ID de membre ou d'indice de matrice.
id_count	long	I	m	Nombre d'éléments de matrice dans les matrices pour les id et les index. Défini à zéro si les matrices pour les id et les index sont NULL.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau B.346.

B.60 Builtin delayfor2

Objet

Le Builtin delayfor2 imprime une chaîne sur l'écran d'application et attend que le temps alloué expire. Des valeurs de VARIABLE peuvent être incorporées dans la chaîne de message avec des informations de formatage.

Structure lexicale

```
delayfor2(duration, prompt, blk_ids, blk_nums, ids, indices, id_count)
```

Les éléments lexicaux du Builtin delayfor2 sont présentés dans le Tableau B.60.

Tableau B.60 – Builtin delayfor2

Nom du paramètre	Type	Direction	Usage	Description
durée	long	I	m	Délai en secondes.
prompt	char[]	I	m	Message à afficher.
blk_ids	array of unsigned long	I	m	ID d'élément des instances de BLOCK_A utilisées dans le message d'invite.
blk_nums	array of unsigned long	I	m	Numéros d'occurrence des instances de BLOCK_A utilisées dans le message d'invite.
ids	array of unsigned long	I	m	ID d'élément d'une instance de VARIABLE utilisée dans le message d'invite.

Nom du paramètre	Type	Direction	Usage	Description
indices	array of unsigned long	I	m	ID de membre ou index de matrice des variables utilisées dans le message d'invite. Défini à zéro s'il n'y a pas d'ID de membre ou d'indice de matrice.
id_count	long	I	m	Nombre d'éléments de matrice dans les matrices pour les id et les index. Défini à zéro si les matrices pour les id et les index sont NULL.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau B.346.

B.61 Builtin DICT_ID

Objet

Le Builtin DICT_ID peut être utilisé pour spécifier un ID de dictionnaire de texte pour les appels pour obtenir stddict_string. Il convertit un nom de dictionnaire de texte en ID. Les Builtin traités par l'outil EDD génèrent l'ID du dictionnaire pour le nom du dictionnaire fourni. La valeur, que l'outil EDD remplace par DICT_ID(nom), est un type de données unsigned long.

Une entrée du dictionnaire de texte est constituée de trois parties. un ID unique, un nom, et une description multilingue. Cette fonction extrait l'ID unique en utilisant le nom du dictionnaire de texte.

Structure lexicale

DICT_ID(name)

Les éléments lexicaux du Builtin DICT_ID sont présentés dans le Tableau B.61.

Tableau B.61 – Builtin DICT_ID

Nom du paramètre	Type	Direction	Usage	Description
name	char[]	I	m	Nom de la chaîne de dictionnaire.
<return>	long	O	m	ID de dictionnaire pour la chaîne nommée.

B.62 Builtin dictionnaire_string

Objet

Le Builtin dictionnaire_string retourne la chaîne du dictionnaire associée au nom donné dans la langue actuelle. Si la chaîne n'est pas disponible dans la langue actuelle, la chaîne en anglais est retournée. Si la chaîne n'est pas trouvée dans le dictionnaire, la fonction retourne "ERROR: Dict String Not Found" (ERREUR: chaîne de dictionnaire introuvable). Le paramètre dict_string_name est le descripteur de la chaîne tel qu'il apparaît dans le dictionnaire.

Structure lexicale

dictionary_string(dict_string_name)

Les éléments lexicaux du Builtin dictionnaire_string sont présentés dans le Tableau B.62.

Tableau B.62 – Builtin dictionnaire_string

Nom du paramètre	Type	Direction	Usage	Description
dict_string_name	reference	l	m	Descripteur de la chaîne tel qu'il apparaît dans le dictionnaire.
<return>	DD_STRING	o	m	Chaîne de dictionnaire.

B.63 Builtin DiffTime**Objet**

Le Builtin DiffTime calcule la différence entre deux heures calendaires.

Structure lexicale

DiffTime(t1, t0)

Les éléments lexicaux du Builtin DiffTime sont présentés dans le Tableau B.63.

Tableau B.63 – Builtin DiffTime

Nom du paramètre	Type	Direction	Usage	Description
t1	time_t	l	m	Heure la plus récente.
t0	time_t	l	m	Heure la moins récente.
<return>	double	o	m	Différence entre t1 et t0 en secondes.

B.64 Builtin discard_on_exit**Objet**

Le Builtin discard_on_exit (rejeter en sortie) définit l'action finale pour rejeter des changements effectués par la méthode sur des valeurs de VARIABLE stockées dans une table de VARIABLE d'application EDD en mémoire cache et qui n'ont pas déjà été envoyées au dispositif. Cela conduit la méthode à rejeter les changements apportés aux valeurs de VARIABLE dans la table de VARIABLE lorsque la méthode se termine.

Lorsqu'une méthode modifie une valeur de VARIABLE dans une application EDD en mémoire cache, elle modifie une copie résidente de cette valeur. Ce changement est effectif uniquement pendant la durée de vie de la méthode. Pour modifier la/les valeur(s) dans le dispositif, la méthode doit envoyer la/les valeur(s) au dispositif ou appeler send_on_exit. Pour enregistrer les valeurs uniquement dans la table de VARIABLE (au lieu de les envoyer au dispositif), appeler le Builtin save_on_exit (enregistrer en sortie).

Après la terminaison de la méthode, les valeurs modifiées dans la table de VARIABLE reprennent les valeurs du dispositif qui existaient avant qu'elles soient modifiées par la méthode.

Toutes les valeurs qui ont été modifiées mais qui n'ont pas été envoyées au dispositif sont traitées en fin de méthode soit en les rejetant, soit en les envoyant au dispositif, ou en enregistrant les modifications d'une manière persistante (mais qui n'affecte pas les valeurs dans le dispositif).

Ce Builtin n'effectue aucune action dans une application EDD non en mémoire cache car les valeurs de paramètre sont écrites directement dans le dispositif.

Si une méthode n'appelle pas `discard_on_exit`, `save_on_exit`, ou `send_on_exit` avant de quitter, les valeurs modifiées par le Builtin sont rejetées comme si le Builtin `discard_on_exit` avait été appelé.

Structure lexicale

`discard_on_exit(VOID)`

L'élément lexical du Builtin `discard_on_exit` est présenté dans le Tableau B.64.

Tableau B.64 – Builtin `discard_on_exit`

Nom du paramètre	Type	Direction	Usage	Description
<return>	VOID	—	—	—

B.65 Builtin DISPLAY

Objet

Le Builtin DISPLAY (afficher) affiche le message spécifié à l'écran, en mettant à jour en continu les valeurs de variable dynamiques utilisées dans la chaîne (voir `PUT_MESSAGE` pour la syntaxe). La valeur de variable continue d'être mise à jour jusqu'à ce que l'opérateur effectue l'accusé de réception.

Structure lexicale

`DISPLAY(prompt)`

Les éléments lexicaux du Builtin DISPLAY sont présentés dans le Tableau B.65.

Tableau B.65 – Builtin DISPLAY

Nom du paramètre	Type	Direction	Usage	Description
prompt	DD_STRING	i	m	Message à afficher.
<return>	VOID	—	—	—

B.66 Builtin display

Objet

Le Builtin display (afficher) affiche le message spécifié à l'écran, en mettant à jour en continu les valeurs de VARIABLE dynamiques utilisées dans la chaîne (voir `PUT_MESSAGE` pour la structure lexicale). La valeur de VARIABLE continue d'être mise à jour jusqu'à ce que l'opérateur effectue l'accusé de réception.

NOTE 1 Il n'est pas garanti que les messages précédemment affichés restent à l'écran.

NOTE 2 Certaines applications EDD effacent l'affichage à chaque appel de cette fonction.

Structure lexicale

`display(prompt, global_var_ids)`

Les éléments lexicaux du Builtin display sont présentés dans le Tableau B.66.

Tableau B.66 – Builtin display

Nom du paramètre	Type	Direction	Usage	Description
prompt	char[]	I	m	Message à afficher.
global_var_ids	array of int	I	m	Matrice contenant des identifiants uniques d'instances de VARIABLE.
<return>	VOID	—	—	—

B.67 Builtin display_bitenum**Objet**

Le Builtin display_bitenum affiche une représentation textuelle d'une VARIABLE BIT_ENUMERATED et attend l'accusé de réception de l'utilisateur.

NOTE 1 Il n'est pas garanti que les messages précédemment affichés restent à l'écran.

NOTE 2 Certaines applications EDD effacent l'affichage à chaque appel de cette fonction.

Structure lexicale

```
display(bitenum_variable)
```

Les éléments lexicaux du Builtin display_bitenum sont présentés dans le Tableau B.67.

Tableau B.67 – Builtin display_bitenum

Nom du paramètre	Type	Direction	Usage	Description
Bitenum_variable	Référence	I	m	VARIABLE BIT_ENUMERATED qui doit être affichée.
<return>	VOID	—	—	—

B.68 Builtin display_builtin_error**Objet**

Le Builtin display_builtin_error affiche sur le dispositif d'affichage de l'application le texte associé à un Builtin_error (erreur) spécifié. Le Builtin display garde le contrôle jusqu'à ce que l'utilisateur accuse réception du message.

NOTE 1 Ce Builtin constitue un moyen de débogage des appels de sous-programmes Builtin qui retournent une erreur. Il convient de ne jamais l'utiliser dans une méthode de production finale, car il convient que les types des erreurs retournées par les Builtin ne soient jamais reçus dans une méthode de débogage total.

NOTE 2 Les codes d'erreur de Builtin peuvent être définis dans un fichier.

Structure lexicale

```
display_builtin_error(error)
```

Les éléments lexicaux du Builtin display_builtin_error sont présentés dans le Tableau B.68.

Tableau B.68 – Builtin display_builtin_error

Nom du paramètre	Type	Direction	Usage	Description
erreur	long	I	m	Code pour une erreur de Builtin.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau B.346.

B.69 Builtin display_comm_error

Objet

Le Builtin display_comm_error affiche la chaîne de texte associée à une erreur de communication sur le dispositif d'affichage de l'application.

NOTE 1 Une erreur de communication de zéro n'est pas considérée comme une erreur de communication.

NOTE 2 Le Builtin ne retourne pas le contrôle à la méthode jusqu'à ce que l'utilisateur accuse réception du message.

En interne, ce Builtin appelle le Builtin get_comm_error_string pour obtenir la chaîne de texte associée et le Builtin get_acknowledgement pour afficher la chaîne de texte associée et attendre l'accusé de réception de l'utilisateur.

Les codes d'erreur de communication sont définis par les consortiums. Leurs chaînes associées peuvent être définies dans un dictionnaire de texte.

Structure lexicale

display_comm_error(error)

Les éléments lexicaux du Builtin display_comm_error sont présentés dans le Tableau B.69.

Tableau B.69 – Builtin display_comm_error

Nom du paramètre	Type	Direction	Usage	Description
erreur	unsigned long	I	m	Code pour une erreur de communication.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau B.346.

B.70 Builtin display_comm_status

Objet

Le Builtin display_comm_status affiche la chaîne associée à la valeur spécifiée de l'octet comm_status_value. Le message reste à l'écran. La chaîne affichée reste à l'écran jusqu'à ce que l'opérateur effectue l'accusé de réception.

NOTE 1 Il n'est pas garanti que les messages précédemment affichés restent à l'écran.

NOTE 2 Certaines applications EDD effacent l'affichage à chaque appel de cette fonction.

Structure lexicale

display_comm_status(comm_status_value)

L'élément lexical du Builtin display_comm_status est présenté dans le Tableau B.70.

Tableau B.70 – Builtin display_comm_status

Nom du paramètre	Type	Direction	Usage	Description
comm_status_value	int	I	m	Code pour une erreur de communication.
<return>	VOID	—	—	—

B.71 Builtin display_device_status

Objet

Le Builtin display_device_status affiche la chaîne associée à la valeur spécifiée de l'octet device_status. La chaîne affichée reste affichée à l'écran jusqu'à ce que l'opérateur effectue l'accusé de réception.

NOTE 1 Il n'est pas garanti que les messages précédemment affichés restent à l'écran.

NOTE 2 Certaines applications EDD effacent l'affichage à chaque appel de cette fonction.

Structure lexicale

```
display_device_status(device_status_value)
```

Les éléments lexicaux du Builtin display_device_status sont présentés dans le Tableau B.71.

Tableau B.71 – Builtin display_device_status

Nom du paramètre	Type	Direction	Usage	Description
device_status_value	int	I	m	Code pour le statut du dispositif.
<return>	VOID	—	—	—

B.72 Builtin display_dynamics

Objet

Le Builtin display_dynamics affiche un message en continu sur le dispositif de sortie de l'application. Le Builtin met à jour les valeurs spécifiées dans le message jusqu'à ce qu'il soit interrompu par l'utilisateur. Jusqu'à ce que l'utilisateur l'interrompe le contrôle n'est pas retourné à la méthode. (Le code d'application recherche l'accusé de réception; le Builtin attend que le code d'application reçoive une intervention de l'utilisateur, la vérifie, et retourne le contrôle au Builtin.)

Des valeurs de VARIABLE peuvent être incorporées dans la chaîne de message avec des informations de formatage. Cet appel peut être effectué pour des variables du dispositif ou variables locales de méthode. Le principal objectif de l'appel est d'afficher des informations de valeur du dispositif que l'utilisateur souhaite examiner pendant une période.

Structure lexicale

```
display_dynamics(prompt, ids, indices, id_count)
```

Les éléments lexicaux du Builtin display_dynamics sont présentés dans le Tableau B.72.

Tableau B.72 – Builtin display_dynamics

Nom du paramètre	Type	Direction	Usage	Description
prompt	char[]	I	m	Message à afficher.
ids	array of unsigned long	I	m	ID d'élément des variables utilisées dans le message d'invite.
indices	array of unsigned long	I	m	ID de membre ou index de matrice des variables utilisées dans le message d'invite. Défini à zéro s'il n'y a pas d'ID de membre ou d'indice de matrice.
id_count	long	I	m	Nombre d'éléments dans les matrices, cela concerne les id et les index. Défini à zéro si les matrices pour les id et les index sont NULL.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau B.346.

B.73 Builtin display_dynamics2**Objet**

Le Builtin display_dynamics2 affiche un message en continu sur le dispositif de sortie de l'application. Le Builtin met à jour les valeurs spécifiées dans le message jusqu'à ce qu'il soit interrompu par l'utilisateur. Jusqu'à ce que l'utilisateur l'interrompe le contrôle n'est pas retourné à la méthode. (Le code d'application recherche l'accusé de réception; le Builtin attend que le code d'application reçoive une intervention de l'utilisateur, la vérifie, et retourne contrôle au Builtin.)

Des valeurs de VARIABLE peuvent être incorporées dans la chaîne de message avec des informations de formatage. Cet appel peut être effectué pour des variables du dispositif ou variables locales de méthode. Le principal objectif de l'appel est d'afficher des informations de valeur du dispositif que l'utilisateur souhaite examiner pendant une période.

Structure lexicale

```
display_dynamics2(prompt, blk_ids, blk_nums, ids, indices, id_count)
```

Les éléments lexicaux du Builtin display_dynamics2 sont présentés dans le Tableau B.73.

Tableau B.73 – Builtin display_dynamics2

Nom du paramètre	Type	Direction	Usage	Description
prompt	char[]	I	m	Message à afficher.
blk_ids	array of unsigned long	I	m	ID d'élément des instances de BLOCK_A utilisées dans le message d'invite.
blk_nums	array of unsigned long	I	m	Numéros d'occurrence des instances de BLOCK_A utilisées dans le message d'invite.
ids	array of unsigned long	I	m	ID d'élément des variables utilisées dans le message d'invite.
indices	array of unsigned long	I	m	ID de membre ou index de matrice des variables utilisées dans le message d'invite. Défini à zéro s'il n'y a pas d'ID de membre ou d'indice de matrice.
id_count	long	I	m	Nombre d'éléments dans les matrices qui concerne les id et les index. Défini à zéro si les matrices pour les id et les index sont NULL.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau B.346.

B.74 Builtin display_message

Objet

Le Builtin display_message affiche un message sur le dispositif de sortie d'application. Il retourne le contrôle à la méthode sans nécessiter un accusé de réception de l'utilisateur.

Les valeurs de VARIABLE de méthode locale du dispositif et les étiquettes pour les VARIABLE du dispositif peuvent être incorporées dans la chaîne d'invite avec des informations de formatage. Cet appel est utilisé pour informer l'utilisateur de changements de statut du système qui ne requièrent aucune action.

Structure lexicale

```
display_message(prompt, ids, indices, id_count)
```

Les éléments lexicaux du Builtin display_message sont présentés dans le Tableau B.74.

Tableau B.74 – Builtin display_message

Nom du paramètre	Type	Direction	Usage	Description
prompt	char[]	l	m	Message à afficher.
ids	array of unsigned long	l	m	ID d'élément des variables utilisées dans le message d'invite.
indices	array of unsigned long	l	m	ID de membre ou index de matrice des variables utilisées dans le message d'invite. Défini à zéro s'il n'y a pas d>ID de membre ou d'indice de matrice.
id_count	long	l	m	Nombre d'éléments dans les matrices, cela concerne les id et les index. Défini à zéro si les matrices pour les id et les index sont NULL.
<return>	long	o	m	Un des codes de retour spécifiés dans le Tableau B.346.

B.75 Builtin display_message2

Objet

Le Builtin display_message2 affiche un message sur le dispositif de sortie d'application. Il retourne le contrôle à la méthode sans nécessiter un accusé de réception de l'utilisateur.

Les valeurs de VARIABLE de méthode locale du dispositif et les étiquettes pour les VARIABLE du dispositif peuvent être incorporées dans la chaîne d'invite avec des informations de formatage. Cet appel est utilisé pour informer l'utilisateur de changements de statut du système qui ne requièrent aucune action.

Structure lexicale

```
display_message2(prompt, blk_ids, blk_nums, ids, indices, id_count)
```

Les éléments lexicaux du Builtin display_message2 sont présentés dans le Tableau B.75.

Tableau B.75 – Builtin display_message2

Nom du paramètre	Type	Direction	Usage	Description
prompt	char[]	I	m	Message à afficher.
blk_ids	array of unsigned long	I	m	ID d'élément des instances de BLOCK_A utilisées dans le message d'invite.
blk_nums	array of unsigned long	I	m	Numéros d'occurrence des instances de BLOCK_A utilisées dans le message d'invite.
ids	array of unsigned long	I	m	ID d'élément des variables utilisées dans le message d'invite.
indices	array of unsigned long	I	m	ID de membre ou index de matrice des variables utilisées dans le message d'invite. Défini à zéro s'il n'y a pas d>ID de membre ou d'indice de matrice.
id_count	long	I	m	Nombre d'éléments dans les matrices, cela concerne les id et les index. Défini à zéro si les matrices pour les id et les index sont NULL.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau B.346.

B.76 Builtin display_response_code

Objet

Le Builtin display_response_code affiche la description d'un code de réponse pour un élément. Le Builtin ne retourne pas le contrôle à la méthode jusqu'à ce que l'utilisateur accuse réception du message.

En interne, ce Builtin appelle le Builtin get_response_code_string pour obtenir la chaîne de texte associée et le Builtin get_acknowledgement pour afficher la chaîne de texte associée et attendre l'accuse de réception de l'utilisateur.

Un code de réponse est une valeur représentant une condition d'erreur spécifique à une application. Une valeur de code de réponse de zéro n'est pas considérée comme un code de réponse d'erreur. Chaque code de réponse et type de code de réponse est défini dans l'EDD.

L'ID de paramètre et l'ID de membre désignent un paramètre, qui peut avoir un code de réponse associé à celui-ci après avoir été lu ou écrit. L'ID de paramètre et l'ID de membre sont spécifiés en utilisant les Builtin ITEM_ID et MEMBER_ID et le nom ou la référence de la variable en question.

Structure lexicale

```
display_response_code(id, member_id, code)
```

Les éléments lexicaux du Builtin display_response_code sont présentés dans le Tableau B.76.

Tableau B.76 – Builtin display_response_code

Nom du paramètre	Type	Direction	Usage	Description
id	unsigned long	I	m	ID d'élément de l'élément qui a généré le code de réponse.
member_id	unsigned long	I	m	ID de membre de l'élément qui a généré le code de réponse.
code	unsigned long	I	m	Code de réponse reçu depuis le Builtin get_response_code.

Nom du paramètre	Type	Direction	Usage	Description
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau B.346.

B.77 Builtin display_response_status

Objet

Le Builtin display_response_status affiche la chaîne associée à la valeur spécifiée de l'octet response_code. La chaîne affichée reste affichée à l'écran jusqu'à ce que l'opérateur effectue l'accusé de réception.

NOTE 1 Il n'est pas garanti que les messages précédemment affichés restent à l'écran.

NOTE 2 Certaines applications EDD effacent l'affichage à chaque appel de cette fonction.

Structure lexicale

```
display_response_status(response_code_value)
```

Les éléments lexicaux du Builtin display_response_status sont présentés dans le Tableau B.77.

Tableau B.77 – Builtin display_response_status

Nom du paramètre	Type	Direction	Usage	Description
response_code_value	int	I	m	Code de réponse à afficher dans une invite.
<return>	VOID	—	—	—

B.78 Builtin display_xmtr_status

Objet

Le Builtin display_xmtr_status affiche la chaîne associée à la valeur spécifiée de la VARIABLE émettrice indiquée. La chaîne affichée reste à l'écran jusqu'à ce que l'opérateur effectue l'accusé de réception.

NOTE 1 Il n'est pas garanti que les messages précédemment affichés restent à l'écran.

NOTE 2 Certaines applications EDD effacent l'affichage à chaque appel de cette fonction.

Structure lexicale

```
display_xmtr_status(xmtr_variable, response_code_value)
```

Les éléments lexicaux du Builtin display_xmtr_status sont présentés dans le Tableau B.78.

Tableau B.78 – Builtin display_xmtr_status

Nom du paramètre	Type	Direction	Usage	Description
xmtr_variable	int	I	m	Référence à une instance de VARIABLE.
response_code_value	int	I	m	Valeur de code de réponse.
<return>	VOID	—	—	—

B.79 Builtin DoubleToByte

Objet

Le Builtin DoubleToByte fournit un vecteur de valeurs de caractère non signé représentant les éléments particuliers d'une valeur double précision.

Structure lexicale

DoubleToByte(in, out1, out2, out3, out4, out5, out6, out7, out8)

Les éléments lexicaux du Builtin DoubleToByte sont présentés dans le Tableau B.79.

Tableau B.79 – Builtin DoubleToByte

Nom du paramètre	Type	Direction	Usage	Description
in	double	I	m	Valeur double précision à convertir.
out1	unsigned char	O	m	Premier élément de la valeur double précision (octet le plus significatif).
out2	unsigned char	O	m	Deuxième élément de la valeur double précision.
out3	unsigned char	O	m	Troisième élément de la valeur double précision.
out4	unsigned char	O	m	Quatrième élément de la valeur double précision.
out5	unsigned char	O	m	Cinquième élément de la valeur double précision.
out6	unsigned char	O	m	Sixième élément de la valeur double précision.
out7	unsigned char	O	m	Septième élément de la valeur double précision.
in	double	I	m	La valeur double précision à convertir.
	char			
out8	unsigned char	O	m	Huitième élément de la valeur à virgule flottante (octet le moins significatif).
<return>	VOID	—	—	—

B.80 Builtin edit_device_value

Objet

Le Builtin edit_device_value affiche une invite avec une valeur du dispositif spécifiée par le param_id et le param_member_id. L'utilisateur peut éditer la valeur affichée.

NOTE Une valeur du dispositif peut résider dans la table de VARIABLE d'une application EDD en mémoire cache, et la valeur ne peut pas être écrite dans le dispositif jusqu'à ce qu'une méthode soit terminée ou la valeur est explicitement envoyée au dispositif par la méthode.

Les valeurs de VARIABLE peuvent être incorporées dans la chaîne de message avec des informations de formatage. L'application édite et vérifie les valeurs entrées pour la valeur du dispositif. Cette édition est basée sur des types d'arithmétique de VARIABLE définis dans l'EDD originale pour chaque VARIABLE. Les attributs EDDL associés aux VARIABLE peuvent comprendre

- HANDLING
- DISPLAY_FORMAT
- EDIT_FORMAT

- MIN_VALUE
- MAX_VALUE

Structure lexicale

```
edit_device_value(prompt, ids, indices, id_count, param_id,
param_member_id)
```

Les éléments lexicaux du Builtin edit_device_value sont présentés dans le Tableau B.80.

Tableau B.80 – Builtin edit_device_value

Nom du paramètre	Type	Direction	Usage	Description
prompt	char[]	I	m	Message à afficher.
ids	array of unsigned long	I	m	ID d'élément des variables utilisées dans le message d'invite.
indices	array of unsigned long	I	m	ID de membre ou index de matrice des variables utilisées dans le message d'invite. Défini à zéro s'il n'y a pas d>ID de membre ou d'indice de matrice.
id_count	long	I	m	Nombre d'éléments dans les matrices, cela concerne les id et les index. Défini à zéro si les matrices pour les id et les index sont NULL.
param_id	unsigned long	I	m	ID d'élément de la variable à modifier.
param_member_id	unsigned long	I	m	ID de membre de la VARIABLE à modifier.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau B.346.

B.81 Builtin edit_device_value2

Objet

Le Builtin edit_device_value2 affiche une invite avec une valeur du dispositif spécifiée par le param_id et le param_member_id. L'utilisateur peut éditer la valeur affichée.

NOTE Une valeur du dispositif peut résider dans la table des variables d'une application EDD en mémoire cache, et la valeur ne peut pas être écrite dans le dispositif jusqu'à ce qu'une méthode se termine ou que la valeur soit explicitement envoyée au dispositif par la méthode.

Des valeurs de variable peuvent être incorporées dans la chaîne de message avec des informations de formatage. L'application édite et vérifie les valeurs entrées pour la valeur du dispositif. Cette édition est basée sur des types d'arithmétique de VARIABLE définis dans l'EDD originale pour chaque VARIABLE. Les attributs EDDL associés aux VARIABLE peuvent comprendre:

- HANDLING
- DISPLAY_FORMAT
- EDIT_FORMAT
- MIN_VALUE
- MAX_VALUE

Structure lexicale

```
edit_device_value2(prompt, blk_ids, blk_nums, ids, indices, id_count,
blk_id, blk_num, param_id, param_member_id)
```

Les éléments lexicaux du Builtin edit_device_value2 sont présentés dans le Tableau B.81.

Tableau B.81 – Builtin edit_device_value2

Nom du paramètre	Type	Direction	Usage	Description
prompt	char[]	I	m	Message à afficher.
blk_ids	array of unsigned long	I	m	ID d'élément des instances de BLOCK_A utilisées dans le message d'invite.
blk_nums	array of unsigned long	I	m	Numéros d'occurrence des instances de BLOCK_A utilisées dans le message d'invite.
ids	array of unsigned long	I	m	ID d'élément des variables utilisées dans le message d'invite.
indices	array of unsigned long	I	m	ID de membre ou index de matrice des variables utilisées dans le message d'invite. Défini à zéro s'il n'y a pas d>ID de membre ou d'indice de matrice.
id_count	long	I	m	Nombre d'éléments dans les matrices, cela concerne les id et les index. Défini à zéro si les matrices pour les id et les index sont NULL.
blk_id	unsigned long	I	m	ID d'élément de l'instance de BLOCK_A à modifier.
blk_num	unsigned long	I	m	Numéro d'occurrence de l'instance de BLOCK_A à modifier.
param_id	unsigned long	I	m	ID d'élément de la VARIABLE à modifier.
param_member_id	unsigned long	I	m	ID de membre de la VARIABLE à modifier.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau B.346.

B.82 Builtin edit_local_value**Objet**

Le Builtin edit_local_value affiche une invite avec le nom de VARIABLE de méthode locale spécifié par le paramètre local_var. Le paramètre local_var est une chaîne de caractères C à terminaison null. L'utilisateur peut éditer la valeur affichée de la VARIABLE de la méthode locale.

Des valeurs de variable peuvent être incorporées dans la chaîne de message avec des informations de formatage. L'application EDD est responsable de la prévention du retour de valeurs non valides en effectuant des vérifications basées sur le type de VARIABLE uniquement. La méthode doit effectuer une vérification d'intervalle sur les valeurs retournées.

Structure lexicale

```
edit_local_value(prompt, ids, indices, id_count, local_var)
```

Les éléments lexicaux du Builtin edit_local_value sont présentés dans le Tableau B.82.

Tableau B.82 – Builtin edit_local_value

Nom du paramètre	Type	Direction	Usage	Description
prompt	char[]	I	m	Message à afficher.
ids	array of unsigned long	I	m	ID d'élément des variables utilisées dans le message d'invite.
indices	array of unsigned	I	m	ID de membre ou index de matrice des variables utilisées dans le message d'invite. Défini à zéro s'il