INTERNATIONAL STANDARD

IEC 61804-2

First edition
2004-05

Function blocks (FB) for process control –

Part 2: Specification of FB concept and Electronic Device Description Language (EDDL)

Reference number
IEC 61804-2:2004(E)

Publication numbering

As from 1 January 1997 all IEC publications are issued with a designation in the 60000 series. For example, IEC 34-1 is now referred to as IEC 60034-1.

Consolidated editions

The IEC is now publishing consolidated versions of its publications. For example, edition numbers 1.0, 1.1 and 1.2 refer, respectively, to the base publication, the base publication incorporating amendment 1 and the base publication incorporating amendments 1 and 2.

Further information on IEC publications

The technical content of IEC publications is kept under constant review by the IEC, thus ensuring that the content reflects current technology. Information relating to this publication, including its validity, is available in the IEC Catalogue of publications (see below) in addition to new editions, amendments and corrigenda. Information on the subjects under consideration and work in progress undertaken by the technical committee which has prepared this publication, as well as the list of publications issued, is also available from the following:

- **IEC Web Site (www.iec.ch)**

- **Catalogue of IEC publications**

  The on-line catalogue on the IEC web site (www.iec.ch/searchpub) enables you to search by a variety of criteria including text searches, technical committees and date of publication. On-line information is also available on recently issued publications, withdrawn and replaced publications, as well as corrigenda.

- **IEC Just Published**

  This summary of recently issued publications (www.iec.ch/online_news/ justpub) is also available by email. Please contact the Customer Service Centre (see below) for further information.

- **Customer Service Centre**

  If you have any questions regarding this publication or need further assistance, please contact the Customer Service Centre:

  Email: custserv@iec.ch
  Tel: +41 22 919 02 11
  Fax: +41 22 919 03 00

INTERNATIONAL STANDARD

IEC 61804-2

First edition
2004-05

Function blocks (FB) for process control –

Part 2: Specification of FB concept and Electronic Device Description Language (EDDL)

International Electrotechnical Commission, 3, rue de Varembé, PO Box 131, CH-1211 Geneva 20, Switzerland
Telephone: +41 22 919 02 11 Telefax: +41 22 919 03 00 E-mail: inmail@iec.ch Web: www.iec.ch

Commission Electrotechnique Internationale
International Electrotechnical Commission
Международная Электротехническая Комиссия

PRICE CODE XH

For price, see current catalogue

CONTENTS

FOREWORD....22
INTRODUCTION....24
1 Scope....25
2 Normative references....26
3 Terms and definitions, abbreviated terms and acronyms, and conventions for lexical structures....27
3.1 Terms and definitions....27
3.2 Abbreviated terms and acronyms....34
3.3 Conventions for lexical structures....34
4 General Function Block (FB) definition and EDD model....36
4.1 Device structure (device model)....36
4.1.1 Device model description....36
4.1.2 FB type....40
4.1.3 FB execution....41
4.1.4 Reference between IEC/PAS 61499-1, IEC/PAS 61499-2 and IEC 61804 models....42
4.1.5 UML specification of the device model....42
4.1.6 Classification of the algorithms....44
4.1.7 Algorithm description....45
4.1.8 Input and output variables and parameter definition....45
4.1.9 Choice of variables and parameters....46
4.1.10 Mode, Status and Diagnosis....46
4.2 Block combinations....46
4.2.1 Measurement channel....46
4.2.2 Actuation channel....47
4.2.3 Application....48
4.3 EDD and EDDL model....49
4.3.1 Overview of EDD and EDDL....49
4.3.2 EDD architecture....49
4.3.3 Concepts of EDD....49
4.3.4 Principles of the EDD development process....49
4.3.5 Interrelations between the lexical structure and formal definitions....50
4.3.6 Builtins....51
4.3.7 Profiles....51
5 Detailed block definition....51
5.1 General....51
5.2 Application FBs....51
5.2.1 Analog Input FB....51
5.2.2 Analog Output FB....53
5.2.3 Discrete Input FB....54
5.2.4 On/Off Actuation (Output) FB Discrete Output FB....56
5.2.5 Calculation FB....57
5.2.6 Control FB....58
5.3 Component FBs....59
5.4 Technology Block....59
5.4.1 Temperature Technology Block....59

5.4.2 Pressure Technology Block 62
5.4.3 Modulating Actuation Technology Block 64
5.4.4 On/Off Actuation Technology Block 66
5.5 Device (Resource) Block 68
5.5.1 Identification 68
5.5.2 Device state 69
5.5.3 Message 71
5.5.4 Initialization 71
5.6 Algorithms common to all blocks 71
5.6.1 Data Input/Data Output status 71
5.6.2 Validity 71
5.6.3 Restart Initialization 71
5.6.4 Fail-safe 72
5.6.5 Remote Cascade Initialization 72
6 FB Environment 73
7 Mapping to System Management 73
8 Mapping to Communication 73
9 Electronic Device Description Language 75
9.1 Overview 75
9.1.1 EDDL features 75
9.1.2 Syntax representation 75
9.1.3 EDD language elements 75
9.1.4 Basic construction elements 75
9.1.5 Common attributes 81
9.1.6 Special elements 81
9.1.7 Rules for instances 81
9.1.8 Rules for list of VARIABLEs 81
9.2 EDD identification information 82
9.2.1 General structure 82
9.2.2 Specific attributes 82
9.3 BLOCK 84
9.3.1 BLOCK_A 84
9.3.2 BLOCK_B 88
9.4 COLLECTION 90
9.4.1 General structure 90
9.4.2 Specific attributes - item-type 90
9.5 COMMAND 91
9.5.1 General structure 91
9.5.2 Specific attributes 92
9.6 CONNECTION 96
9.6.1 General structure 96
9.6.2 Specific attribute - APPINSTANCE 96
9.7 DOMAIN 97
9.7.1 General structure 97
9.7.2 Specific attribute - HANDLING 97
9.8 EDIT_DISPLAY 98
9.8.1 General structure 98
9.8.2 Specific attributes 98

9.9 IMPORT 100
9.9.1 General structure 100
9.9.2 Specific attributes – attribute-redefinition 102
9.10 LIKE 108
9.11 MENU 108
9.11.1 General structure 108
9.11.2 Specific attributes 109
9.11.3 Sequence diagrams for actions 114
9.12 METHOD 118
9.12.1 General structure 118
9.12.2 Specific attributes 118
9.13 PROGRAM 119
9.13.1 General structure 119
9.13.2 Specific attributes - ARGUMENT 119
9.14 RECORD 120
9.15 REFERENCE_ARRAY 120
9.15.1 General structure 120
9.15.2 Specific attributes - ELEMENTS 121
9.16 Relations 121
9.16.1 General structure 121
9.16.2 REFRESH 121
9.16.3 UNIT 122
9.16.4 WRITE_AS_ONE 122
9.17 RESPONSE_CODES 122
9.18 VALUE_ARRAY 123
9.18.1 General structure 123
9.18.2 Specific attributes 123
9.19 VARIABLE 124
9.19.1 General structure 124
9.19.2 Specific attributes 125
9.20 VARIABLE_LIST 138
9.21 Common attributes 138
9.21.1 DEFINITION 138
9.21.2 HELP 139
9.21.3 LABEL 139
9.21.4 MEMBERS 139
9.21.5 RESPONSE_CODES 140
9.22 Output redirection (OPEN and CLOSE) 141
9.23 Conditional expression 141
9.24 Referencing 142
9.24.1 Referencing an EDD instance 142
9.24.2 Referencing members of a RECORD 142
9.24.3 Referencing elements of a VALUE_ARRAY 143
9.24.4 Referencing members of a COLLECTION 143
9.24.5 Referencing elements of a REFERENCE_ARRAY 144
9.24.6 Referencing members of a VARIABLE_LISTS 144
9.24.7 Referencing elements of BLOCK_A PARAMETERS 145
9.24.8 Referencing elements of BLOCK_A PARAMETER_LISTS 145
9.24.9 Referencing BLOCK_A CHARACTERISTICS 146

9.25 Strings 146
9.25.1 Specifying a string as a string literal 146
9.25.2 Specifying a string as a string variable 146
9.25.3 Specifying a string as an enumeration value 146
9.25.4 Specifying a string as a dictionary reference 147
9.25.5 Referencing HELP and LABEL attributes of EDD instances 147
9.25.6 String operations 147
9.25.7 Prompt String Formats 148
9.26 Expression 148
9.26.2 Primary expressions 148
9.26.3 Unary expressions 150
9.26.4 Binary expressions 150
9.27 Text dictionary 153
10 Conformance statement 154
Annex A (informative) Parameter description 155
Annex B (normative) IEC 61804 Conformance Declaration 160
Annex C (normative) EDDL Formal Definition 161
C.1 EDDL Preprocessor 161
C.1.1 General structure 161
C.1.2 Directives 161
C.1.3 Predefined macros 164
C.1.4 NEWLINE characters 164
C.1.5 Comments 164
C.2 Conventions 164
C.2.1 Integer constants 164
C.2.2 Floating point constants 165
C.2.3 String literals 165
C.2.4 Using language codes in string constants 166
C.3 Operators 166
C.4 Keywords 167
C.5 Terminals 169
C.6 letter (letter|digit|_)*Formal EDDL syntax 169
C.6.1 General 169
C.6.2 EDD identification information 169
C.6.3 BLOCK_A and BLOCK_B 171
C.6.4 COLLECTION 173
C.6.5 COMMAND 173
C.6.6 CONNECTION 177
C.6.7 DOMAIN 177
C.6.8 EDIT_DISPLAY 178
C.6.9 IMPORT 179
C.6.10 LIKE 180
C.6.11 MENU 182
C.6.12 METHOD 185
C.6.13 PROGRAM 185
C.6.14 RECORDS 186
C.6.15 REFERENCE_ARRAY 186
C.6.16 Relations 187

C.6.17 RESPONSE_CODES 188
C.6.18 VALUE_ARRAY 189
C.6.19 VARIABLE 189
C.6.20 VARIABLE_LIST 197
C.6.21 Common attributes 197
C.6.22 OPEN, CLOSE 199
C.6.23 Expression 199
C.6.24 C-Grammer 201
C.6.25 Redefinition 204
C.6.26 References 216
Annex D (normative) EDDL Builtin Library 218
D.1 General 218
D.2 Conventions for Builtin descriptions 218
D.3 Builtin abort 218
D.4 Builtin abort_on_all_comm_errors 219
D.5 Builtin ABORT_ON_ALL_COMM_STATUS 219
D.6 Builtin ABORT_ON_ALL_DEVICE_STATUS 220
D.7 Builtin ABORT_ON_ALL_RESPONSE_CODES 220
D.8 Builtin abort_on_all_response_codes 221
D.9 Builtin abort_on_comm_error 221
D.10 Builtin ABORT_ON_COMM_ERROR 222
D.11 Builtin ABORT_ON_COMM_STATUS 222
D.12 Builtin ABORT_ON_DEVICE_STATUS 223
D.13 Builtin ABORT_ON_NO_DEVICE 223
D.14 Builtin ABORT_ON_RESPONSE_CODE 224
D.15 Builtin abort_on_response_code 225
D.16 Builtin ACKNOWLEDGE 225
D.17 Builtin acknowledge 225
D.18 Builtin add_abort_method (version A) 226
D.19 Builtin add_abort_method (version B) 226
D.20 Builtin assign 227
D.21 Builtin assign_double 227
D.22 Builtin assign_float 228
D.23 Builtin assign_int 228
D.24 Builtin assign_var 228
D.25 Builtin atof 229
D.26 Builtin atoi 229
D.27 Builtin dassign 229
D.28 Builtin Date_to_DayOfMonth 230
D.29 Builtin Date_to_Month 230
D.30 Builtin Date_to_Year 231
D.31 Builtin DELAY 231
D.32 Builtin delay 231
D.33 Builtin DELAY_TIME 232

D.34 Builtin delayfor 232
D.35 Builtin DICT_ID 233
D.36 Builtin discard_on_exit 233
D.37 Builtin display 234
D.38 Builtin display_builtin_error 234
D.39 Builtin display_comm_error 234
D.40 Builtin display_comm_status 235
D.41 Builtin display_device_status 235
D.42 Builtin display_dynamics 236
D.43 Builtin display_message 236
D.44 Builtin display_response_code 237
D.45 Builtin display_response_status 238
D.46 Builtin display_xmtr_status 238
D.47 Builtin edit_device_value 238
D.48 Builtin edit_local_value 239
D.49 Builtin ext_send_command 240
D.50 Builtin ext_send_command_trans 240
D.51 Builtin fail_on_all_comm_errors 241
D.52 Builtin fail_on_all_response_codes 242
D.53 Builtin fail_on_comm_error 242
D.54 Builtin fail_on_response_code 243
D.55 Builtin fassign 243
D.56 Builtin fgetval 244
D.57 Builtin float_value 244
D.58 Builtin fsetval 244
D.59 Builtin ftoa 245
D.60 Builtin fvar_value 245
D.61 Builtin get_acknowledgement 245
D.62 Builtin get_comm_error 246
D.63 Builtin get_comm_error_string 247
D.64 Builtin get_date 247
D.65 Builtin get_date_value 247
D.66 Builtin get_dds_error 248
D.67 Builtin GET_DEV_VAR_VALUE 249
D.68 Builtin get_dev_var_value 249
D.69 Builtin get_dictionary_string 250
D.70 Builtin get_double 250
D.71 Builtin get_double_value 251
D.72 Builtin get_float 251
D.73 Builtin get_float_value 252
D.74 Builtin GET_LOCAL_VAR_VALUE 252
D.75 Builtin get_local_var_value 253
D.76 Builtin get_more_status 253

D.77 Builtin get_resolve_status 254
D.78 Builtin get_response_code 254
D.79 Builtin get_response_code_string 255
D.80 Builtin get_signed 255
D.81 Builtin get_signed_value 256
D.82 Builtin get_status_code_string 256
D.83 Builtin get_status_string 257
D.84 Builtin get_stddict_string 257
D.85 Builtin get_string 258
D.86 Builtin get_string_value 259
D.87 Builtin GET_TICK_COUNT 259
D.88 Builtin get_unsigned 259
D.89 Builtin get_unsigned_value 260
D.90 Builtin iassign 260
D.91 Builtin igetval 261
D.92 Builtin IGNORE_ALL_COMM_STATUS 261
D.93 Builtin IGNORE_ALL_DEVICE_STATUS 262
D.94 Builtin IGNORE_ALL_RESPONSE_CODES 262
D.95 Builtin IGNORE_COMM_ERROR 263
D.96 Builtin IGNORE_COMM_STATUS 263
D.97 Builtin IGNORE_DEVICE_STATUS 264
D.98 Builtin IGNORE_NO_DEVICE 264
D.99 Builtin IGNORE_RESPONSE_CODE 265
D.100 Builtin int_value 265
D.101 Builtin is_NaN 266
D.102 Builtin isetval 266
D.103 Builtin ITEM_ID 266
D.104 Builtin itoa 267
D.105 Builtin ivar_value 267
D.106 Builtin lassign 267
D.107 Builtin lgetval 268
D.108 Builtin LOG_MESSAGE 268
D.109 Builtin long_value 268
D.110 Builtin lsetval 269
D.111 Builtin lvar_value 269
D.112 Builtin MEMBER_ID 269
D.113 Builtin method_abort 270
D.114 Builtin ObjectReference 270
D.115 Builtin process_abort 270
D.116 Builtin put_date 271
D.117 Builtin put_date_value 271
D.118 Builtin put_double 272
D.119 Builtin put_double_value 272

D.120 Builtin put_float 273
D.121 Builtin put_float_value 273
D.122 Builtin PUT_MESSAGE 274
D.123 Builtin put_message 274
D.124 Builtin put_signed 275
D.125 Builtin put_signed_value 275
D.126 Builtin put_string 276
D.127 Builtin put_string_value 276
D.128 Builtin put_unsigned 277
D.129 Builtin put_unsigned_value 278
D.130 Builtin READ_COMMAND 278
D.131 Builtin read_value 279
D.132 Builtin remove_abort_method (version A) 279
D.133 Builtin remove_abort_method (version B) 280
D.134 Builtin remove_all_abort_methods 280
D.135 Builtin resolve_array_ref 280
D.136 Builtin resolve_block_ref 281
D.137 Builtin resolve_param_list_ref 281
D.138 Builtin resolve_param_ref 282
D.139 Builtin resolve_record_ref 282
D.140 Builtin retry_on_all_comm_errors 283
D.141 Builtin RETRY_ON_ALL_COMM_STATUS 283
D.142 Builtin RETRY_ON_ALL_DEVICE_STATUS 284
D.143 Builtin RETRY_ON_ALL_RESPONSE_CODES 284
D.144 Builtin retry_on_all_response_codes 285
D.145 Builtin RETRY_ON_COMM_ERROR 285
D.146 Builtin retry_on_comm_error 286
D.147 Builtin RETRY_ON_COMM_STATUS 286
D.148 Builtin RETRY_ON_DEVICE_STATUS 287
D.149 Builtin RETRY_ON_NO_DEVICE 287
D.150 Builtin RETRY_ON_RESPONSE_CODE 288
D.151 Builtin retry_on_response_code 288
D.152 Builtin rspcode_string 289
D.153 Builtin save_on_exit 289
D.154 Builtin save_values 290
D.155 Builtin SELECT_FROM_LIST 290
D.156 Builtin select_from_list 291
D.157 Builtin select_from_menu 291
D.158 Builtin send 292
D.159 Builtin send_all_values 292
D.160 Builtin send_command 293
D.161 Builtin send_command_trans 293
D.162 Builtin send_on_exit 294

D.163 Builtin send_trans 294
D.164 Builtin send_value 295
D.165 Builtin SET_NUMBER_OF_RETRIES 296
D.166 Builtin To_Date_and_Time 296
D.167 Builtin VARID 296
D.168 Builtin vassign 297
D.169 Builtin WRITE_COMMAND 297
D.170 Builtin XMTR_ABORT_ON_ALL_COMM_STATUS 297
D.171 Builtin XMTR_ABORT_ON_ALL_DEVICE_STATUS 298
D.172 Builtin XMTR_ABORT_ON_ALL_RESPONSE_CODES 298
D.173 Builtin XMTR_ABORT_ON_COMM_ERROR 299
D.174 Builtin XMTR_ABORT_ON_COMM_STATUS 299
D.175 Builtin XMTR_ABORT_ON_DATA 300
D.176 Builtin XMTR_ABORT_ON_DEVICE_STATUS 300
D.177 Builtin XMTR_ABORT_ON_NO_DEVICE 301
D.178 Builtin XMTR_ABORT_ON_RESPONSE_CODE 301
D.179 Builtin XMTR_IGNORE_ALL_COMM_STATUS 302
D.180 Builtin XMTR_IGNORE_ALL_DEVICE_STATUS 302
D.181 Builtin XMTR_IGNORE_ALL_RESPONSE_CODES 303
D.182 Builtin XMTR_IGNORE_COMM_ERROR 303
D.183 Builtin XMTR_IGNORE_COMM_STATUS 304
D.184 Builtin XMTR_IGNORE_DEVICE_STATUS 304
D.185 Builtin XMTR_IGNORE_NO_DEVICE 305
D.186 Builtin XMTR_IGNORE_RESPONSE_CODE 305
D.187 Builtin XMTR_RETRY_ON_ALL_DEVICE_STATUS 306
D.188 Builtin XMTR_RETRY_ON_ALL_RESPONSE_CODE 306
D.189 Builtin XMTR_RETRY_ON_ALL_RESPONSE_CODES 307
D.190 Builtin XMTR_RETRY_ON_COMM_ERROR 307
D.191 Builtin XMTR_RETRY_ON_COMM_STATUS 308
D.192 Builtin XMTR_RETRY_ON_DATA 308
D.193 Builtin XMTR_RETRY_ON_DEVICE_STATUS 309
D.194 Builtin XMTR_RETRY_ON_NO_DEVICE 309
D.195 Builtin XMTR_RETRY_ON_RESPONSE_CODE 310
D.196 Builtin YearMonthDay_to_Date 310
D.197 Builtins Return Codes 310
Annex E (informative) EDD Example 312
Annex F (normative) Profiles of EDDL and Builtins 331
F.1 Profile of EDDL and Builtins 331
F.2 Profiles for PROFIBUS 332
F.2.1 EDDL profile 332
F.2.2 Builtin profile 334
F.2.3 EDDL Formal Definition profile 337
F.3 Profiles for Fieldbus Foundation ® 338

F.3.1 EDDL profile 338
F.3.2 Builtin profile 341
F.3.3 EDDL Formal Definition profile 344
F.4 Profiles for HART® Communication Foundation (HCF) 345
F.4.1 EDDL profile 345
F.4.2 Builtin profile 347
F.4.3 EDDL Formal Definition profile 351
F.5 Data types 351
F.5.1 METHOD DEFINITIONS data types 351
F.5.2 Coding of data DATE 353
F.5.3 Coding of data DATE_AND_TIME 353
F.5.4 Coding of data DURATION 353
F.5.5 Coding of data TIME 353
F.5.6 Coding of data TIME_VALUE 354
F.5.7 Coding of PACKED_ASCII (6-BIT ASCII) DATA FORMAT 354
Bibliography 356

Figure 1 – Position of the IEC 61804 series related to other standards and products 26
Figure 3 – FB structure may be distributed between devices (according to IEC/PAS 61499–1) 37
Figure 4 – IEC 61804 FBs can be implemented in different devices 38
Figure 5 – General components of devices 38
Figure 7 – IEC 61804 block overview (graphical representation not normative) 40
Figure 8 – UML class diagram of the device model 43
Figure 9 – Measurement process signal flow 47
Figure 10 – Actuation process signal flow 47
Figure 11 – Application process signal flow 48
Figure 12 – EDD generation process 50
Figure 13 – Analog Input FB 52
Figure 14 – Analog Output FB 53
Figure 15 – Discrete input FB 55
Figure 16 – Discrete Output FB 56
Figure 17 – Calculation FB 57
Figure 18 – Control FB 58
Figure 19 – Temperature Technology Block 60
Figure 20 – Pressure Technology Block 63
Figure 21 – Modulating actuation technology block 65
Figure 22 – On/Off Actuation Technology Block 67
Figure 23 – Harel state chart 70
Figure 24 – Application structure of ISO OSI Reference Model 73
Figure 25 – Client/Server relationship in terms of OSI Reference Model 74
Figure 26 – Mapping of IEC 61804 FBs to APOs 74
Figure 27 – BLOCK_A 76
Figure 28 – COLLECTION 76
Figure 29 – COMMAND 77

Figure 30 – DOMAIN 77
Figure 31 – EDIT_DISPLAY 77
Figure 32 – LIKE 78
Figure 33 – MENU 78
Figure 34 – PROGRAM 78
Figure 35 – RECORD 79
Figure 36 – REFERENCE_ARRAY 79
Figure 37 – REFRESH 79
Figure 38 – UNIT 79
Figure 39 – WRITE_AS_ONE 80
Figure 40 – VALUE_ARRAY 80
Figure 41 – VARIABLE 80
Figure 42 – VARIABLE_LIST 81
Figure 43 – EDDL import mechanisms 100
Figure 44 – MENU activation (ACCESS OFFLINE) 114
Figure 45 – Action performed after a new value is entered 114
Figure 46 – Action performed after all VARIABLE inputs of the MENU are accepted (ACCESS OFFLINE) 115
Figure 47 – Method execution 115
Figure 48 – MENU activation (ACCESS ONLINE) 116
Figure 49 – Cyclic reading of dynamic VARIABLEs (ACCESS ONLINE) 117
Figure 50 – Action performed after all VARIABLE inputs of the MENU are accepted (ACCESS ONLINE) 117
Figure 51 – Time for read and write operation 137
Figure E.1 – Example of an operator screen using EDD 312

Table 1 – Field attribute descriptions 35
Table 2 – References of model elements 42
Table 3 – Variables and parameter description template 45
Table 4 – Example of temperature sensors of Sensor_Type 61
Table 5 – Device status state table 69
Table 6 – Device status transition table 70
Table 7 – DD_REVISION attribute 82
Table 8 – DEVICE_REVISION attribute 83
Table 9 – DEVICE_TYPE attribute 83
Table 10 – EDD_PROFILE attribute 83
Table 11 – EDD_VERSION attribute 84
Table 12 – MANUFACTURER attribute 84
Table 13 – MANUFACTURER_EXT attribute 84
Table 14 – BLOCK_A attributes 85
Table 15 – CHARACTERISTIC attribute 85
Table 16 – PARAMETER attributes 86
Table 17 – COLLECTION_ITEMS attribute 86
Table 18 – EDIT_DISPLAY_ITEMS attribute 86

Table 19 – MENU_ITEMS attribute 86
Table 20 – METHOD_ITEMS attribute 87
Table 21 – PARAMETER_LISTS attributes 87
Table 22 – REFERENCE_ARRAY_ITEMS attribute 87
Table 23 – REFRESH_ITEMS attribute 88
Table 24 – UNIT_ITEMS attribute 88
Table 25 – WRITE_AS_ONE_ITEMS attribute 88
Table 26 – BLOCK_B attributes 89
Table 27 – NUMBER attributes 89
Table 28 – TYPE attributes 89
Table 29 – COLLECTION attributes 90
Table 30 – item-type 90
Table 31 – COMMAND attributes 91
Table 32 – OPERATION attribute 92
Table 33 – TRANSACTION attributes 92
Table 34 – REPLY and REQUEST attributes 93
Table 35 – INDEX attribute 94
Table 36 – BLOCK_B attribute 94
Table 37 – NUMBER attribute 95
Table 38 – SLOT attribute 95
Table 39 – CONNECTION attribute 95
Table 40 – HEADER attribute 96
Table 41 – MODULE attribute 96
Table 42 – CONNECTION attribute 96
Table 43 – APPINSTANCE attribute 97
Table 44 – DOMAIN attributes 97
Table 45 – HANDLING attribute 97
Table 46 – EDIT_DISPLAY attributes 98
Table 47 – EDIT_ITEMS attribute 98
Table 48 – DISPLAY_ITEM attributes 99
Table 49 – POST_EDIT_ACTIONS, PRE_EDIT_ACTIONS attribute 99
Table 50 – Importing Device Description 101
Table 51 – Redefinition attributes 102
Table 52 – Redefinition rules for BLOCK_A attributes 102
Table 53 – Redefinition rules for BLOCK_B attributes 103
Table 54 – Redefinition rules for COLLECTION attributes 103
Table 55 – Redefinition rules for COMMAND attributes 103
Table 56 – Redefinition rules for CONNECTION attributes 104
Table 57 – Redefinition rules for DOMAIN attributes 104
Table 58 – Redefinition rules for EDIT_DISPLAY attributes 104
Table 59 – Redefinition rules for MENU attributes 104
Table 60 – Redefinition rules for METHOD attributes 105
Table 61 – Redefinition rules for PROGRAM attributes 105

Table 62 – Redefinition rules for RECORD attributes 105
Table 63 – Redefinition rules for REFERENCE_ARRAY attributes 106
Table 64 – Redefinition rules for RESPONSE_CODES attributes 106
Table 65 – Redefinition rules for VALUE_ARRAY attributes 106
Table 66 – Redefinition rules for VARIABLE attributes 107
Table 67 – Redefinition rules for VARIABLE_LIST attributes 107
Table 68 – LIKE attributes 108
Table 69 – MENU attribute 108
Table 70 – ITEMS attribute 109
Table 71 – ACCESS attribute 110
Table 72 – ENTRY attribute 110
Table 73 – POST_EDIT_ACTIONS, PRE_EDIT_ACTIONS, POST_READ_ACTIONS, PRE_READ_ACTIONS, POST_WRITE_ACTIONS, PRE_WRITE_ACTIONS attributes 110
Table 74 – PURPOSE attribute 112
Table 75 – ROLE attribute 113
Table 76 – STYLE attribute 113
Table 77 – VALIDITY attributes 114
Table 78 – METHOD attributes 118
Table 79 – ACCESS attributes 118
Table 80 – VALIDITY attributes 119
Table 81 – PROGRAM attributes 119
Table 82 – ARGUMENT attribute 120
Table 83 – RECORD attributes 120
Table 84 – REFERENCE_ARRAY attribute 120
Table 85 – ELEMENTS attribute 121
Table 86 – REFRESH attributes 121
Table 87 – UNIT attributes 122
Table 88 – WRITE_AS_ONE attribute 122
Table 89 – RESPONSE_CODES attributes 123
Table 90 – VALUE_ARRAY attributes 123
Table 91 – NUMBER_OF_ELEMENTS attribute 124
Table 92 – TYPE attribute 124
Table 93 – VARIABLE attributes 125
Table 94 – CLASS attributes 126
Table 95 – TYPE attributes 127
Table 96 – DOUBLE, FLOAT, INTEGER, UNSIGNED_INTEGER attributes 128
Table 97 – BIT_ENUMERATED attributes 130
Table 98 – status–class attributes 131
Table 99 – ALL, AO, DV, TV attributes 132
Table 100 – Enumerated types attributes 132
Table 101 – Index type attributes 133
Table 102 – Object reference type attribute 133
Table 103 – DEFAULT_REFERENCE attributes 133

Table 104 – String types attributes 134
Table 105 – CONSTANT_UNIT attribute 135
Table 106 – HANDLING attribute 135
Table 107 – POST_EDIT_ACTIONS, PRE_EDIT_ACTIONS, POST_READ_ACTIONS, PRE_READ_ACTIONS, POST_WRITE_ACTIONS, PRE_WRITE_ACTIONS attributes 135
Table 108 – READ/WRITE_TIMEOUT attributes 137
Table 109 – STYLE attribute 137
Table 110 – VALIDITY attributes 138
Table 111 – VARIABLE_LIST attributes 138
Table 112 – DEFINITION attributes 139
Table 113 – HELP attribute 139
Table 114 – LABEL attribute 139
Table 116 – RESPONSE_CODES attribute 140
Table 117 – OPEN and CLOSE attributes 141
Table 118 – IF, SELECT conditional 142
Table 119 – Referencing an EDD instance 142
Table 120 – Referencing elements of RECORD 143
Table 121 – Referencing elements of VALUE_ARRAY 143
Table 122 – Referencing members of COLLECTION 143
Table 123 – Referencing members of REFERENCE_ARRAY 144
Table 124 – Referencing members of VARIABLE_LISTS 144
Table 125 – Referencing members of a BLOCK_A PARAMETERS 145
Table 126 – Referencing members of BLOCK_A PARAMETER_LISTS 145
Table 127 – Referencing BLOCK_A CHARACTERISTICS 146
Table 128 – String as a string literal 146
Table 129 – String as a string variable 146
Table 130 – String as an enumeration value 147
Table 131 – String as a dictionary reference 147
Table 132 – Referencing HELP and LABEL attributes of EDD instances 147
Table 133 – String operation 148
Table 134 – Format specifier 148
Table 135 – Primary expressions 149
Table 136 – Attribute values of VARIABLEs 149
Table 137 – Unary expressions 150
Table 138 – Multiplicative operators 151
Table 139 – Additive operators 151
Table 140 – Shift operators 151
Table 141 – Relational operators 151
Table 142 – Equality operators 152
Table 143 – Text dictionary attributes 153
Table A.1 – Parameter description 155
Table B.1 – Conformance (sub)clause selection table 160
Table B.2 – Contents of (sub)clause selection tables 160

Table C.1 – Conventions for Integer Constants 165
Table C.2 – Using escape sequences in string literals 166
Table C.3 – Using language codes in string literals 166
Table C.4 – EDDL Operators 167
Table C.5 – EDDL Keywords 167
Table D.1 – Format for the Builtins lexical element tables 218
Table D.2 – Contents of the lexical element table 218
Table D.3 – Builtin abort 219
Table D.4 – Builtin abort_on_all_comm_errors 219
Table D.5 – Builtin ABORT_ON_ALL_COMM_STATUS 220
Table D.6 – Builtin ABORT_ON_ALL_DEVICE_STATUS 220
Table D.7 – Builtin ABORT_ON_ALL_RESPONSE_CODES 221
Table D.8 – Builtin abort_on_all_response_codes 221
Table D.9 – Builtin abort_on_comm_error 221
Table D.10 – Builtin ABORT_ON_COMM_ERROR 222
Table D.11 – Builtin ABORT_ON_COMM_STATUS 222
Table D.12 – Builtin ABORT_ON_DEVICE_STATUS 223
Table D.13 – Builtin ABORT_ON_NO_DEVICE 223
Table D.14 – Builtin ABORT_ON_RESPONSE_CODE 224
Table D.15 – Builtin abort_on_response_code 225
Table D.16 – Builtin ACKNOWLEDGE 225
Table D.17 – Builtin acknowledge 226
Table D.18 – Builtin add_abort_method 226
Table D.19 – Builtin add_abort_method 227
Table D.20 – Builtin assign 227
Table D.21 – Builtin assign_double 228
Table D.22 – Builtin assign_float 228
Table D.23 – Builtin assign_int 228
Table D.24 – Builtin assign_var 229
Table D.25 – Builtin atof 229
Table D.26 – Builtin atoi 229
Table D.27 – Builtin dassign 230
Table D.28 – Builtin Date_to_DayOfMonth 230
Table D.29 – Builtin Date_to_Month 230
Table D.30 – Builtin Date_to_Year 231
Table D.31 – Builtin DELAY 231
Table D.32 – Builtin delay 232
Table D.33 – Builtin DELAY_TIME 232
Table D.34 – Builtin delayfor 232
Table D.35 – Builtin DICT_ID 233
Table D.36 – Builtin discard_on_exit 234
Table D.37 – Builtin display 234
Table D.38 – Builtin display_builtin_error 234

Table D.39 – Builtin display_comm_error 235
Table D.40 – Builtin display_comm_status 235
Table D.41 – Builtin display_device_status 236
Table D.42 – Builtin display_dynamics 236
Table D.43 – Builtin display_message 237
Table D.44 – Builtin display_response_code 237
Table D.45 – Builtin display_response_status 238
Table D.46 – Builtin display_xmtr_status 238
Table D.47 – Builtin edit_device_value 239
Table D.48 – Builtin edit_local_value 240
Table D.49 – Builtin ext_send_command 240
Table D.50 – Builtin ext_send_command_trans 241
Table D.51 – Builtin fail_on_all_comm_errors 241
Table D.52 – Builtin fail_on_all_response_codes 242
Table D.53 – Builtin fail_on_comm_error 243
Table D.54 – Builtin fail_on_response_code 243
Table D.55 – Builtin fassign 244
Table D.56 – Builtin fgetval 244
Table D.57 – Builtin float_value 244
Table D.58 – Builtin fsetval 245
Table D.59 – Builtin ftoa 245
Table D.60 – Builtin fvar_value 245
Table D.61 – Builtin get_acknowledgement 246
Table D.62 – Builtin get_comm_error 246
Table D.63 – Builtin get_comm_error_string 247
Table D.64 – Builtin get_date 247
Table D.65 – Builtin get_date_value 248
Table D.66 – Builtin get_dds_error 248
Table D.67 – Builtin GET_DEV_VAR_VALUE 249
Table D.68 – Builtin get_dev_var_value 249
Table D.69 – Builtin get_dictionary_string 250
Table D.70 – Builtin get_double 250
Table D.71 – Builtin get_double_value 251
Table D.72 – Builtin get_float 251
Table D.73 – Builtin get_float_value 252
Table D.74 – Builtin GET_LOCAL_VAR_VALUE 252
Table D.75 – Builtin get_local_var_value 253
Table D.76 – Builtin get_more_status 253
Table D.77 – Builtin get_resolve_status 254
Table D.78 – Builtin get_response_code 255
Table D.79 – Builtin get_response_code_string 255
Table D.80 – Builtin get_signed 256
Table D.81 – Builtin get_signed_value 256

Table D.82 – Builtin get_status_code_string 257
Table D.83 – Builtin get_status_string 257
Table D.84 – Builtin get_stddict_string 258
Table D.85 – Builtin get_string 258
Table D.86 – Builtin get_string_value 259
Table D.87 – Builtin GET_TICK_COUNT 259
Table D.88 – Builtin get_unsigned 260
Table D.89 – Builtin get_unsigned_value 260
Table D.90 – Builtin iassign 261
Table D.91 – Builtin igetval 261
Table D.92 – Builtin IGNORE_ALL_COMM_STATUS 262
Table D.93 – Builtin IGNORE_ALL_DEVICE_STATUS 262
Table D.94 – Builtin IGNORE_ALL_RESPONSE_CODES 263
Table D.95 – Builtin IGNORE_COMM_ERROR 263
Table D.96 – Builtin IGNORE_COMM_STATUS 264
Table D.97 – Builtin IGNORE_DEVICE_STATUS 264
Table D.98 – Builtin IGNORE_NO_DEVICE 265
Table D.99 – Builtin IGNORE_RESPONSE_CODE 265
Table D.100 – Builtin int_value 265
Table D.101 – Builtin is_NaN 266
Table D.102 – Builtin isetval 266
Table D.103 – Builtin ITEM_ID 266
Table D.104 – Builtin itoa 267
Table D.105 – Builtin ivar_value 267
Table D.106 – Builtin lassign 267
Table D.107 – Builtin lgetval 268
Table D.108 – Lexical elements of Builtin LOG_MESSAGE 268
Table D.109 – Builtin long_value 268
Table D.110 – Builtin lsetval 269
Table D.111 – Builtin lvar_value 269
Table D.112 – Builtin MEMBER_ID 269
Table D.113 – Builtin method_abort 270
Table D.114 – Builtin process_abort 270
Table D.115 – Builtin process_abort 271
Table D.116 – Builtin put_date 271
Table D.117 – Builtin put_date_value 272
Table D.118 – Builtin put_double 272
Table D.119 – Builtin put_double_value 273
Table D.120 – Builtin put_float 273
Table D.121 – Builtin put_float_value 274
Table D.122 – Builtin PUT_MESSAGE 274
Table D.123 – Builtin put_message 275
Table D.124 – Builtin put_signed 275

Table D.125 – Builtin put_signed_value 276
Table D.126 – Builtin put_string 276
Table D.127 – Builtin put_string_value 277
Table D.128 – Builtin put_unsigned 278
Table D.129 – Builtin put_unsigned_value 278
Table D.130 – Lexical elements of Builtin READ_COMMAND 279
Table D.131 – Builtin read_value 279
Table D.132 – Builtin remove_abort_method 279
Table D.133 – Builtin remove_abort_method 280
Table D.134 – Builtin remove_all_abort_methods 280
Table D.135 – Builtin resolve_array_ref 281
Table D.136 – Builtin resolve_block_ref 281
Table D.137 – Builtin resolve_param_list_ref 282
Table D.138 – Builtin resolve_parm_ref 282
Table D.139 – Builtin resolve_record_ref 283
Table D.140 – Builtin retry_on_all_comm_errors 283
Table D.141 – Builtin RETRY_ON_ALL_COMM_STATUS 284
Table D.142 – Builtin RETRY_ON_ALL_DEVICE_STATUS 284
Table D.143 – Builtin RETRY_ON_ALL_RESPONSE_CODES 285
Table D.144 – Builtin retry_on_all_response_codes 285
Table D.145 – Builtin RETRY_ON_COMM_ERROR 286
Table D.146 – Builtin retry_on_comm_error 286
Table D.147 – Builtin RETRY_ON_COMM_STATUS 287
Table D.148 – Builtin RETRY_ON_DEVICE_STATUS 287
Table D.149 – Builtin RETRY_ON_NO_DEVICE 288
Table D.150 – Builtin RETRY_ON_RESPONSE_CODE 288
Table D.151 – Builtin retry_on_response_code 289
Table D.152 – Builtin rspcode_string 289
Table D.153 – Builtin save_on_exit 290
Table D.154 – Builtin save_values 290
Table D.155 – Builtin SELECT_FROM_ LIST 290
Table D.156 – Builtin select_from_list 291
Table D.157 – Builtin select_from_menu 292
Table D.158 – Builtin send 292
Table D.159 – Builtin send_all_values 293
Table D.160 – Builtin send_command 293
Table D.161 – Builtin send_command_trans 294
Table D.162 – Builtin send_on_exit 294
Table D.163 – Builtin send_trans 295
Table D.164 – Builtin send_value 295
Table D.165 – Builtin SET_NUMBER_OF_RETRIES 296
Table D.166 – Builtin To_Date_and_Time 296
Table D.167 – Builtin VARID 297

Table D.168 – Builtin vassign 297
Table D.169 – Builtin WRITE_COMMAND 297
Table D.170 – Builtin XMTR_ABORT_ON_ALL_COMM_STATUS 298
Table D.171 – Builtin XMTR_ABORT_ON_ALL_DEVICE_STATUS 298
Table D.172 – Builtin XMTR_ABORT_ON_ALL_RESPONSE_CODES 299
Table D.173 – Builtin XMTR_ABORT_ON_COMM_ERROR 299
Table D.174 – Builtin XMTR_ABORT_ON_COMM_STATUS 300
Table D.175 – Builtin XMTR_ABORT_ON_DATA 300
Table D.176 – Builtin XMTR_ABORT_ON_DEVICE_STATUS 301
Table D.177 – Builtin XMTR_ABORT_ON_NO_DEVICE 301
Table D.178 – Builtin XMTR_ABORT_ON_RESPONSE_CODE 302
Table D.179 – Builtin XMTR_IGNORE_ALL_COMM_STATUS 302
Table D.180 – Builtin XMTR_IGNORE_ALL_DEVICE_STATUS 303
Table D.181 – Builtin XMTR_IGNORE_ALL_RESPONSE_CODES 303
Table D.182 – Builtin XMTR_IGNORE_COMM_ERROR 304
Table D.183 – Builtin XMTR_IGNORE_COMM_STATUS 304
Table D.184 – Builtin XMTR_IGNORE_DEVICE_STATUS 305
Table D.185 – Builtin XMTR_IGNORE_NO_DEVICE 305
Table D.186 – Builtin XMTR_IGNORE_RESPONSE_CODE 306
Table D.187 – Builtin XMTR_RETRY_ON_ALL_DEVICE_STATUS 306
Table D.188 – Builtin XMTR_RETRY_ON_ALL_RESPONSE_CODE 307
Table D.189 – Builtin XMTR_RETRY_ON_ALL_RESPONSE_CODES 307
Table D.190 – Builtin XMTR_RETRY_ON_COMM_ERROR 307
Table D.191 – Builtin XMTR_RETRY_ON_COMM_STATUS 308
Table D.192 – Builtin XMTR_RETRY_ON_DATA 308
Table D.193 – Builtin XMTR_RETRY_ON_DEVICE_STATUS 309
Table D.194 – Builtin XMTR_RETRY_ON_NO_DEVICE 309
Table D.195 – Builtin XMTR_RETRY_ON_RESPONSE_CODE 310
Table D.196 – Builtin YearMonthDay_to_Date 310
Table D.197 – Contents of the return codes description table 311
Table D.198 – Return Code Description 311
Table F.1 – Profile selection tables 331
Table F.2 – EDDL Formal Definition profile tables 331
Table F.3 – Contents of selection tables 331
Table F.4 – EDDL element selection for PROFIBUS 332
Table F.5 – Builtin profile for PROFIBUS 334
Table F.7 – EDDL element selection for Fieldbus Foundation 338
Table F.8 – Builtin profile for Fieldbus Foundation 341
Table F.9 – EDDL element selection for HCF 344
Table F.10 – Builtin profile for HCF 345
Table F.11 – METHOD DEFINITIONS data types 347
Table F.12 – VARIABLE TYPES 351
Table F.13 – DATE coding 351

Table F.14 – DATE_AND_TIME coding 352
Table F.15 – DURATION coding..... 353
Table F.16 – TIME coding 353
Table F.17 – TIME_VALUE coding 353
Table F.18 – PACKED_ASCII coding..... 354

INTERNATIONAL ELECTROTECHNICAL COMMISSION

FUNCTION BLOCKS (FB) FOR PROCESS CONTROL –

Part 2: Specification of FB concept and 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 provides no marking procedure to indicate its approval and cannot be rendered responsible for any equipment declared to be in conformity with an IEC Publication.
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.

The International Electrotechnical Commission (IEC) draws attention to the fact that it is claimed that compliance with this document may involve the use of patents

U.S. Patent No. 5,333,114

U.S. Patent No. 5,485,400

U.S. Patent No. 5,825,664

U.S. Patent No. 5,909,368

U.S. Patent Pending No. 08/916,178

Australian Patent No. 638507

Canadian Patent No. 2,066,743

European Patent No. 0495001

Validated in:

UK – Patent No. 0495001

France – Patent No. 0495001

Germany – Patent No. 69032954.7

Netherlands – Patent No. 0495001

Japan Patent No. 3137643

IEC take no position concerning the evidence, validity and scope of this patent right. The holder of this patent right has assured the IEC that he is willing to negotiate licenses under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of this patent right is registered with IEC. Information may be obtained from:

Fieldbus Foundation,

9390 Research Boulevard, Suite II-250,

Austin, Texas, USA 78759,

Attention: President.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those identified above. IEC shall not be held responsible for identifying any or all such patent rights.

The International Standard IEC 61804-2 has been prepared by subcommittee 65C: Digital communications, of IEC technical committee 65: Industrial-process measurement and control.

This first edition of IEC 61804-2 cancels and replaces the IEC/PAS 61804-2 (2002).

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

FDIS	Report on voting
65C/324/FDIS	65C/337/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.

Parts of this standard are derived from "Fieldbus Foundation Specification FF-890 rev. 1.5 (undated)" and "Fieldbus Foundation Specification FF-900 rev. 1.4 (dated 1999-06-29)" and are used with permission of the Fieldbus Foundation[1].

Parts of this standard are derived from "HART Device Description Language Specification, rev. 11.0 (August 5, 1996)" and "HART Device Description Language Method Builtins Library, rev. 10.1 (August 5, 1996)" and are used with the permission of the HART Communication Foundation[2].

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

IEC 61804 consists of the following parts:

Part 1: *Overview of system aspects* (Technical Report)

Part 2: *Specification of FB concept and Electronic Device Description Language (EDDL)*

The committee has decided that the contents of this publication will remain unchanged until 2006. At this date, the publication will be

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

---

1 For additional information on Fieldbus Foundation, please contact: The Fieldbus Foundation, 9390 Research Boulevard, Austin, Texas 78759, USA. Tel: +1 512 794 8890. URL: www.fieldbusfoundation.org.

2 For additional information on HART Communication Foundation, please contact: HART Communication Foundation, 9390 Research Boulevard, Suite I-350, Austin, Texas 78759, USA. Tel: +1 512 794 0369. URL: www.hartcomm.org.

INTRODUCTION

This part of IEC 61804 provides conceptual Function Block specifications, which can be mapped to specific communication systems and their accompanying definitions by industrial groups and specify the Electronic Device Description Language (EDDL).

The EDDL and the device-related Electronic Device Description (EDD) is intended for use in industrial automation applications. These applications 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 standard specifies a generic language for describing the properties of automation system components. The specified language 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.

The language is called "Electronic Device Description Language (EDDL)" and is used to create "Electronic Device Descriptions (EDD)". These EDDs may be 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 C, but it is possible to use the semantic model also with different syntaxes.

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

FUNCTION BLOCKS (FB) FOR PROCESS CONTROL –

Part 2: Specification of FB concept and Electronic Device Description Language (EDDL)

1 Scope

This Part of IEC 61804 is applicable to Function Blocks (FB) for process control and specifies the Electronic Device Description Language (EDDL).

This standard specifies FB by using the result of harmonization work as regards several elements.

a) The device model which defines the components of an IEC 61804-2 conformant device.

b) Conceptual specifications of FBs for measurement, actuation and processing. This includes general rules for the essential features to support control, whilst avoiding details which stop innovation as well as specialization for different industrial sectors.

c) The Electronic Device Description (EDD) technology, which enables the integration of real product details using the tools of the engineering life cycle.

This standard defines a subset of the requirements of IEC 61804-1 (hereafter referred to as Part 1) only, while Part 1 describes requirements for a distributed system.

The conformance statement in Annex B, which covers the conformance declaration, is related to this standard only. Requirements of Part 1 are not part of these conformance declarations.

The standardization work for FB was carried out by harmonizing the description of concepts of existing technologies. It results in an abstract level that allowed the definition of the common features in a unique way. This abstract vision is called here the conceptual FB specification and mapped to specific communication systems and their accompanying definitions by the industrial groups. This standard is also based on the abstract definitions of IEC/PAS 61499-1.

NOTE This standard can be mapped to ISO 15745-1.

The EDDL is ready for use and fills the gap between the conceptual FB specification 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.

There are solutions on the market today, which fulfil the requirements of this standard and show how the conceptual specification is implemented in a given technology. New technologies will need to find equivalent solutions (see Figure 4).

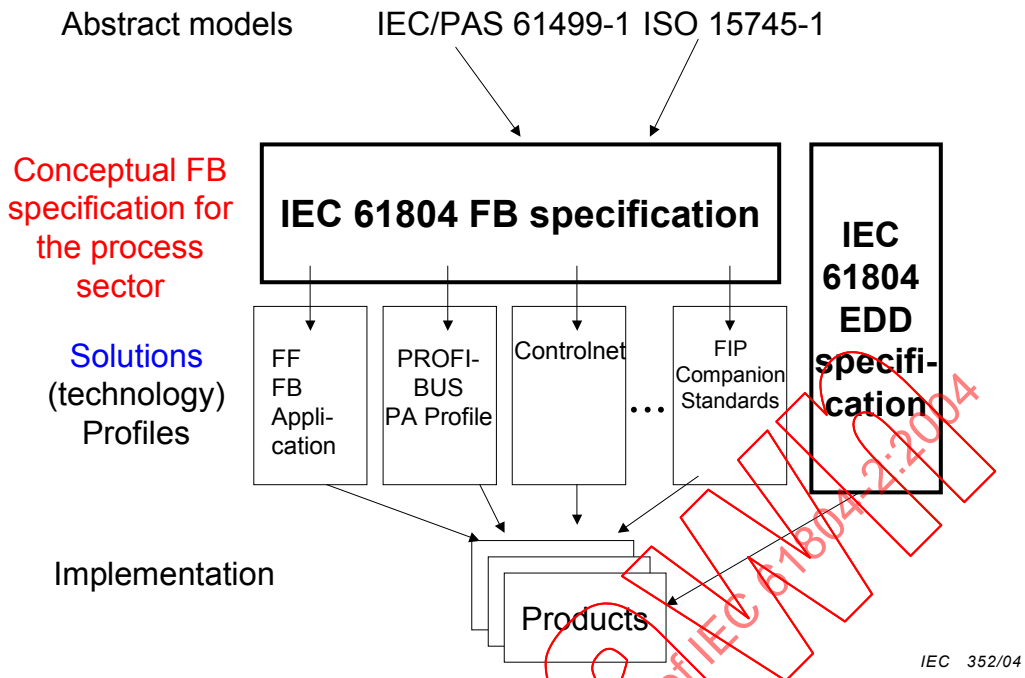


Figure 1 – Position of the IEC 61804 series related to other standards and products

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 60050-351:1998, *International Electrotechnical Vocabulary (IEV) – Part 351: Automatic control*

IEC 60584-1, *Thermocouples – Part 1: Reference tables*

IEC 61131-3:2003, *Programmable controllers – Part 3: Programming languages*

IEC 61158:2003 (all parts), *Digital data communications for measurement and control – Fieldbus for use in industrial control systems*

IEC/PAS 61499-1:2000, *Function blocks for industrial-process measurement and control systems – Part 1: Architecture*

IEC/PAS 61499-2:2001, *Function blocks for industrial-process measurement and control systems – Part 2: Software tools requirements*

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

ISO/IEC 2022:1994, *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 7498-1:1994, *Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*

ISO/IEC 8859-1, *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, *Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Part 1: Architecture and Basic Multilingual Plane*

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

3 Terms and definitions, abbreviated terms and acronyms, and conventions for lexical structures

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.

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
application
software functional unit that is specific to the solution of a problem in industrial-process measurement and control

NOTE An application may be distributed among *resources*, and may communicate with other applications.

3.1.3
application function block
FB which has no input or output to the process

3.1.4
attribute
property or characteristic of an entity, for instance, the version identifier of a FB type specification

[IEC/PAS 61499-1]

NOTE The formal description of Attributes is part of the solution profiles to achieve domain specific interoperability. IEC 61804 defines the general rules to define the attributes and specifies the EDDL to describe attributes, which may be described in solution profiles.

3.1.5
Builtin
predefined subroutine for communication and display executed by the EDD application

3.1.6
component function block
FB instance which is used in the specification of an algorithm of a composite FB type

NOTE A component FB can be a FB or a composite FB type.

3.1.7
composite FB type
FB type whose algorithms is expressed entirely in terms of interconnected component FBs and variables

[IEC/PAS 61499-1]

3.1.8
configuration (of a system or device)
step in system design: selecting *functional units*, assigning their locations and defining their interconnections

[IEC/PAS 61499-1]

3.1.9
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]

3.1.10
data connection
association established between functional units for conveyance of data

[IEC/PAS 61499-1]

3.1.11
data input
interface of a FB which receives data from a data connection

[IEC/PAS 61499-1]

3.1.12
data output
interface of a FB, which supplies data to a data connection

[IEC/PAS 61499-1]

3.1.13
data type
a set of values together with a set of permitted operations

[ISO 2382 series]

3.1.14
device
independent physical entity capable of performing one or more specified functions in a particular context and delimited by its interfaces

[IEC/PAS 61499-1]

3.1.15
Device block
FB which has no input and no output

3.1.16
device management application
application whose primary function is the management of a multiple resources within a device

[IEC/PAS 61499-1]

3.1.17
EDD application
program using the EDD, or any translated form, which offers functionality like communication representation, data representation, graphical representation, etc.

3.1.18
EDDL processor
processor or program, which translates the EDD into an executable form that can be processed by an EDD application

3.1.19
EDDL profile
selection of the supported elements of the EDDL lexical structure including the syntax definitions for a number of specific consortia

3.1.20
Electronic Device Description Language (EDDL)
methodology for describing parameter(s) of an automation system component

3.1.21
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)

3.1.22
Electronic Device Description Source (EDDS)
ASCII file containing a specific device description

3.1.23
Electronic Device Description Technology (EDDT)
technology which includes the EDD development process, the EDD usage and the involved tool chain

3.1.24
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

3.1.25
Electronic Device Description Compiler
compiler which translates the EDD source in an internal format that is used by the EDD Interpreter

3.1.26
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.

3.1.27
entity
particular thing, such as a person, place, process, object, concept, association, or event

[IEC/PAS 61499-1]

3.1.28
event
instantaneous occurrence that is significant to scheduling the execution of an algorithm

[IEC/PAS 61499-1]

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

3.1.29
exception
event that causes suspension of normal execution

[IEC/PAS 61499-1]

[ANSI/IEEE Std 729-1983]

3.1.30
function
specific purpose of an entity or its characteristic action

[IEC/PAS 61499-1]

3.1.31
functional unit
entity of hardware or software, or both, capable of accomplishing a specified purpose

[ISO/AFNOR Dictionary of Computer Science]

3.1.32
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/PAS 61499-1]

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

3.1.33
function block diagram
network in which the nodes are function block instances, variables, literals, and events

NOTE This is not the same as the *function block diagram* defined in IEC 61131-3.

[IEC/PAS 61499-1]

3.1.34
hardware
physical equipment, as opposed to programs, procedures, rules and associated documentation

[ISO/AFNOR Dictionary of Computer Science]

3.1.35
implementation
development phase in which the hardware and software of a system become operational

[IEC/PAS 61499-1]

3.1.36
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 a *FB*, as defined in IEC 61131-3, is an *input variable*.

[IEC/PAS 61499-1]

3.1.37
instance
functional unit comprising an individual, named entity with the attributes of a defined type

[IEC/PAS 61499-1]

3.1.38
instance name
identifier associated with, and designating, an instance

[IEC/PAS 61499-1]

3.1.39
instantiation
creation of an instance of a specified type

[IEC/PAS 61499-1]

3.1.40
interface
shared boundary between two *functional units*, defined by functional characteristics, signal characteristics, or other characteristics as appropriate

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

3.1.41
internal variable
variable whose value is used or modified by one or more operations of a FB but is not supplied by a data input or to a data output

[IEC/PAS 61499-1]

3.1.42
invocation
process of initiating the execution of the sequence of operations specified in an algorithm

[IEC/PAS 61499-1]

3.1.43
management function block
FB whose primary function is the management of applications within a resource

[IEC/PAS 61499-1]

3.1.44
mapping
set of values having defined correspondence with the quantities or values of another set

[ISO/AFNOR Dictionary of Computer Science]

3.1.45
model
representation of a real world process, device, or concept

[IEC/PAS 61499-1]

3.1.46
operation
well-defined action that, when applied to any permissible combination of known entities, produces a new entity

[ISO/AFNOR Dictionary of Computer Science]

3.1.47
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 a *FB*, as defined in IEC 61131-3, is an *output variable*.

[IEC/PAS 61499-1]

3.1.48
parameter
variable that is given a constant value for a specified application and that may denote the application

[ISO/AFNOR Dictionary of Computer Science]

3.1.49
preprocessor
part of the run-time environment that transforms the information given in an EDD

3.1.50
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

3.1.51
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.

3.1.52
resource management application
application whose primary function is the management of a single resource

[IEC/PAS 61499-1]

3.1.53
service
functional capability of a resource, which can be modeled by a sequence of service primitives

[IEC/PAS 61499-1]

3.1.54
software
intellectual creation comprising the programs, procedures, rules and any associated documentation pertaining to the operation of a system

[IEC/PAS 61499-1]

3.1.55
system
set of interrelated elements considered in a defined context as a whole and separated from its environment

[IEC 60050-351:1998, 11-01]

NOTE 1 Such elements may be both material objects and concepts as well as the results thereof (e.g. 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.

3.1.56
technology block
FB, which has at least one input or one output to the process

3.1.57
text dictionary

collection of multilingual or other texts within the EDD

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

3.1.58
type
software element, which specifies the common attributes shared by all instances of the type

[IEC/PAS 61499-1]

3.1.59
type name
identifier associated with, and designating, a type

[IEC/PAS 61499-1]

3.1.60
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/PAS 61499-1]

3.2 Abbreviated terms and acronyms

The terms in IEC 60050-351:1998 apply partially.

ADU	Analog Digital Unit
AFB	Application Function Block
ANSI	American National Standard Institut:
ANSI C	American National Standard Institute for the programming language C (see ISO/IEC 9899)
AP	Application Process
ASCII	American Standard Code for Information Interchange (see ISO/IEC 10646-1)
ASN.1	Abstract Lexical Structure Notation 1
BNF	Backus Naur Format
CFB	Component Function Block
DAU	Digital Analog Unit
DD	Device Description
DTD	Data Type Definition
EDD	Electronic Device Description
EDDI	Electronic Device Description Interpreter
EDDL	Electronic Device Description Language
EUC	Extended Unit Code (see ISO/IEC 2022:1994)
FB	Function Block
FBD	Function Block Diagram
FMS	Fieldbus Message Specification
HMI	Human Machine Interface
HTML	Hypertext Mark-up Language
I/O	Input/Output
IAM	Intelligent Actuation and Measurement
ID	Identifier
mA	Milliampere
NOAH	Network Oriented Application Harmonization
OSI	Open Systems Interconnection
P&ID	Piping and Instrument Diagram
PDU	Protocol Data Unit
SM	System Management
TB	Technology Block
UML	Unified Modelling Language
wao	Write as one

3.3 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.

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 market 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 9.23). The expression <exp> refers only to the fields within the preceding curved brackets (). Usage of conditional expressions is optional.

4 General Function Block (FB) definition and EDD model

4.1 Device structure (device model)

4.1.1 Device model description

FBs are encapsulations of variables and their processing algorithms. The variables and algorithms are those required by the design of the process and its control system.

NOTE FBs can be derived from the diagram in Figure 2.

FBs perform the application (measurement, actuation, control and monitoring) by connecting their data inputs and data outputs.

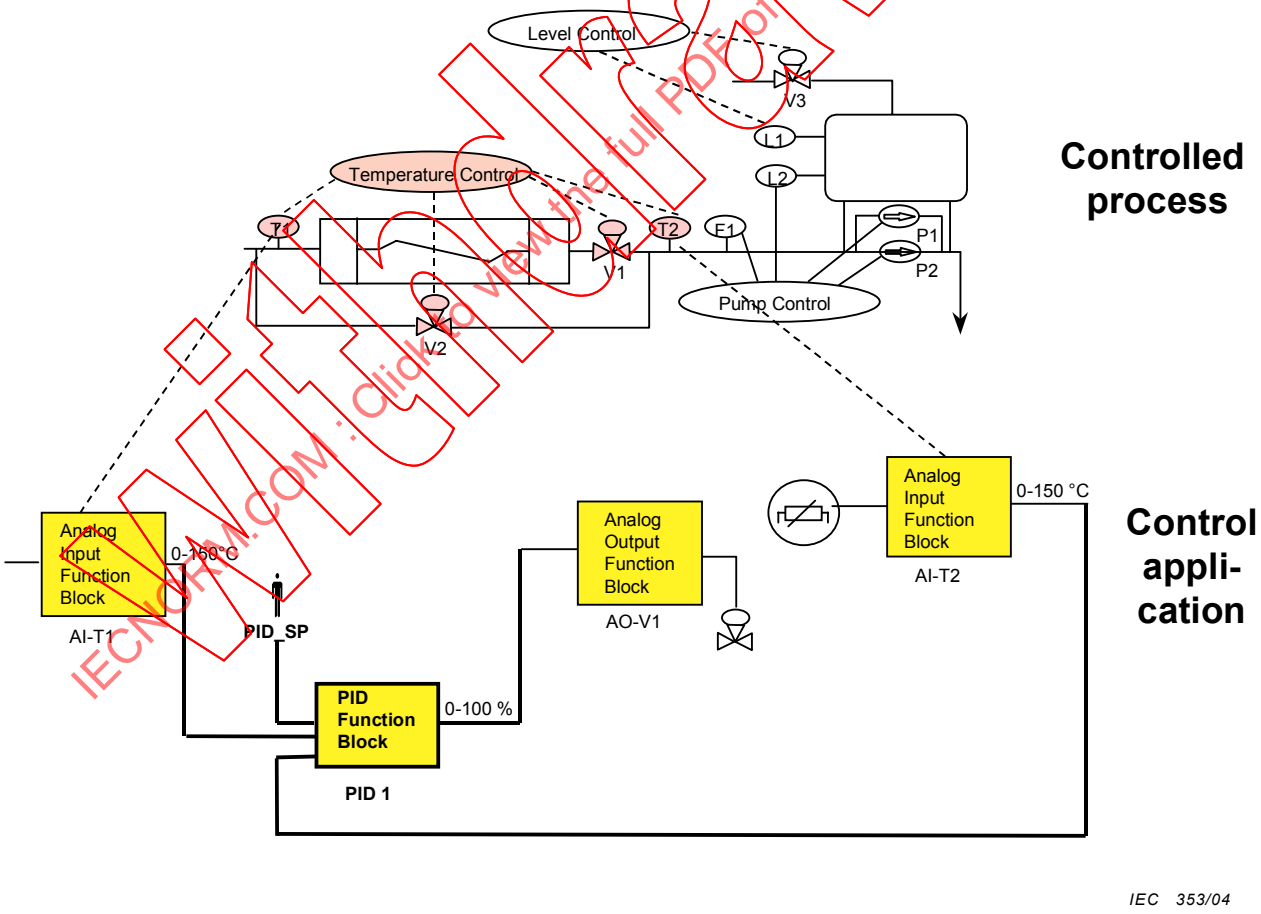


Figure 2 – FB structure is derived out of the process (P&ID view)

The devices are connected via a communication network or a hierarchy of communication networks.

NOTE The application may be distributed among several devices; see, for example, Figure 3.

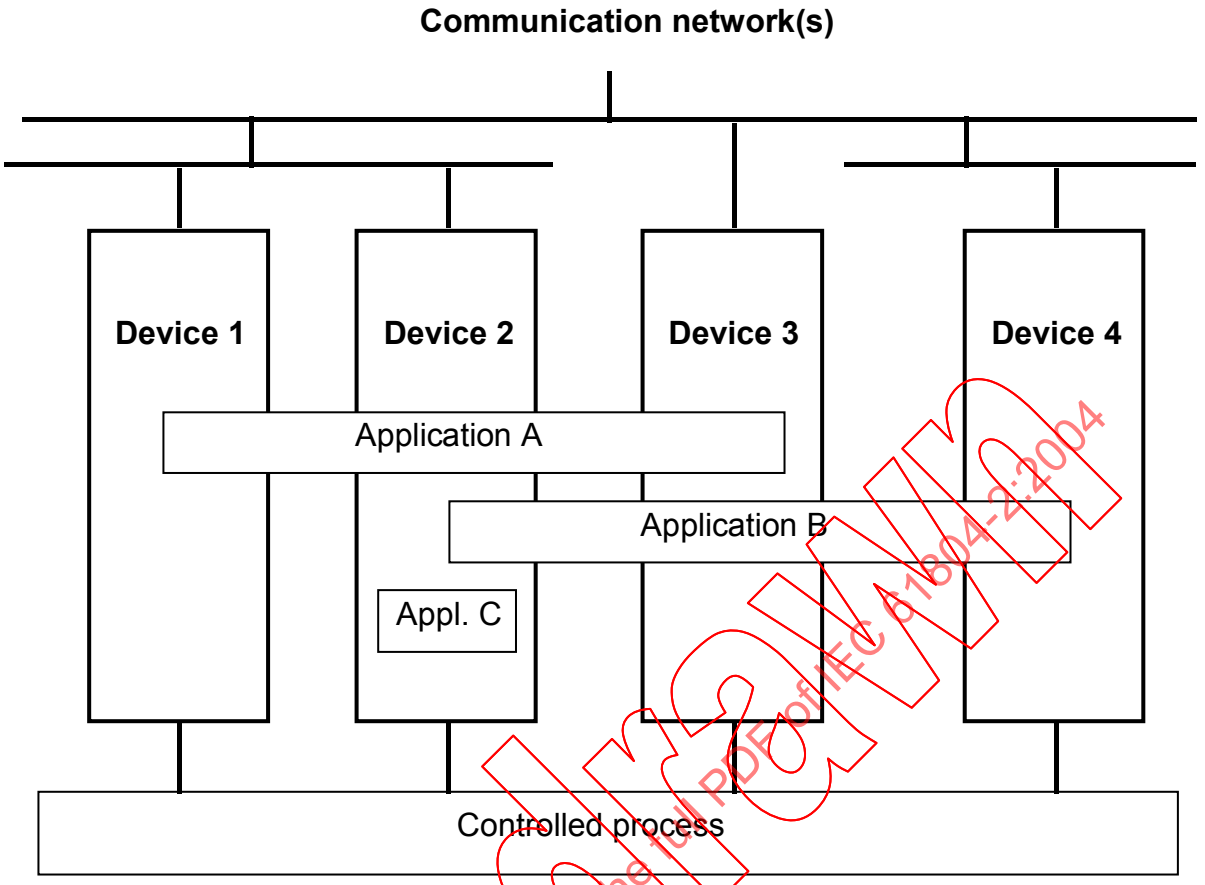


Figure 3 – FB structure may be distributed between devices (according to IEC/PAS 61499–1)

The FBs resulting from the design of the control system are abstract representations.

NOTE 1 These can be implemented in different ways in different device types, see Figure 4. FBs can be implemented, for example, in field devices, programmable logic controller, visualization stations and device descriptions.

Additionally, other applications such as system engineering and supervisory system have to handle or interact with the FBs.

NOTE 2 Algorithms defined for a FB in the conceptual model are not necessarily mapped one-to-one to the device; they can be mapped to the device, a proxy or a supervisory station if the current technology does not solve it in the device.

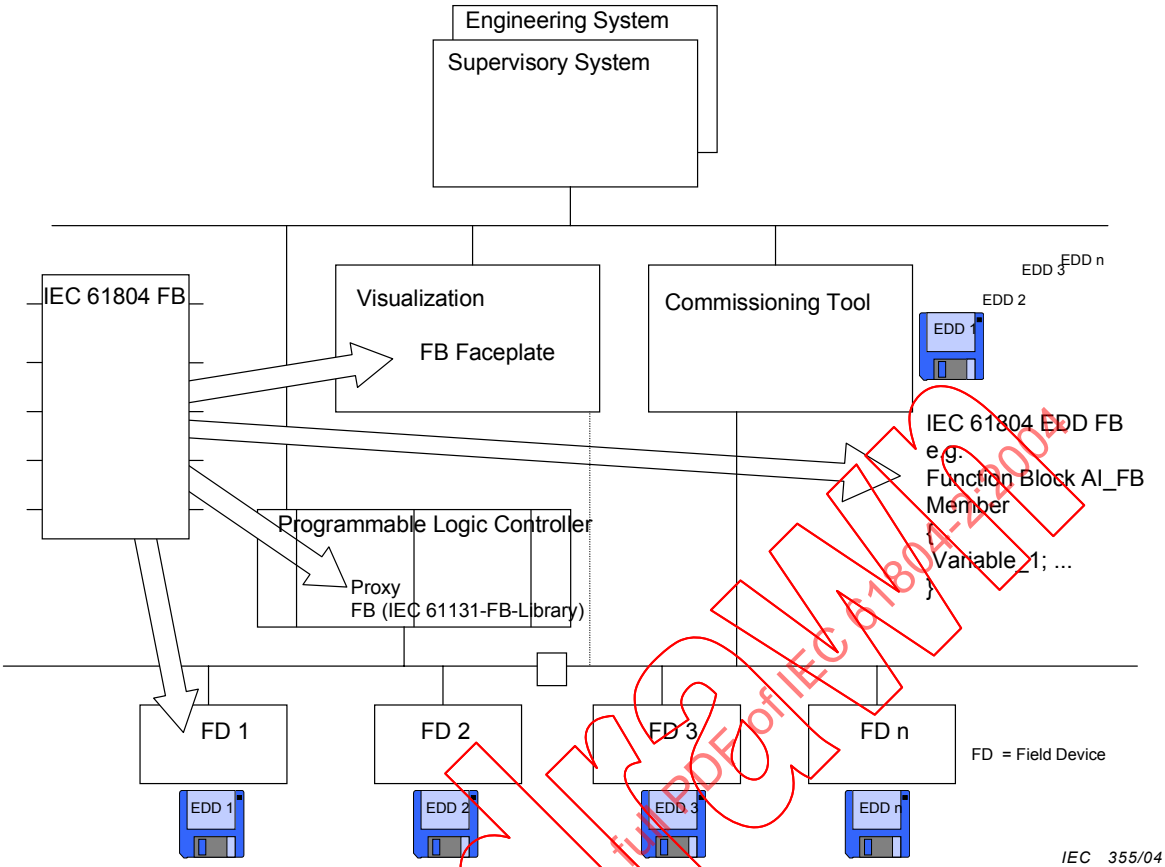


Figure 4 – IEC 61804 FBs can be implemented in different devices

For the purpose of this standard, devices implement algorithms derived out of the design of the controlled process in terms of FBs. The devices are hardware and software modular (see for example Figure 5). The components of devices are Modules, Blocks, Variables and Algorithms. There are defined relations between the components that are specified in the UML class diagram below (see Figure 8).

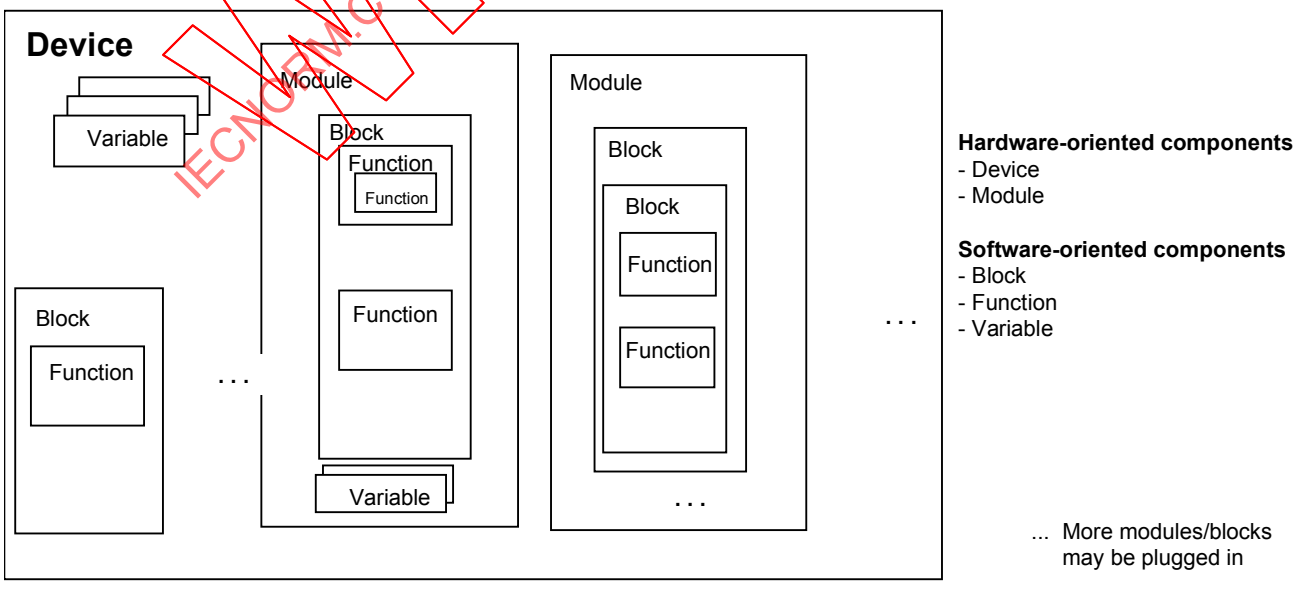


Figure 5 – General components of devices

For the purpose of this standard, there are different block types (see Figure 6), which encapsulate specific functionality of devices performing an automation application. The Technology Block represents the process attachment of a device. It contains the measurement or actuation principles of a device. The technology block is composed of acquisition or output and transformation parts. The application FB (hereafter called FB) contains application-related signal processing, such as scaling, alarm detection or control and calculation. Component FBs may perform mathematical and logical processing with specific additional exception handling procedures such as not-allowed parameter values. They shall be encapsulated within composite FBs.

The Device Block represents the resource of the device that contains information and function about the device itself, the operation system of the device and the device hardware. The device shall have an interface to the communication system and may have system management functionalities.

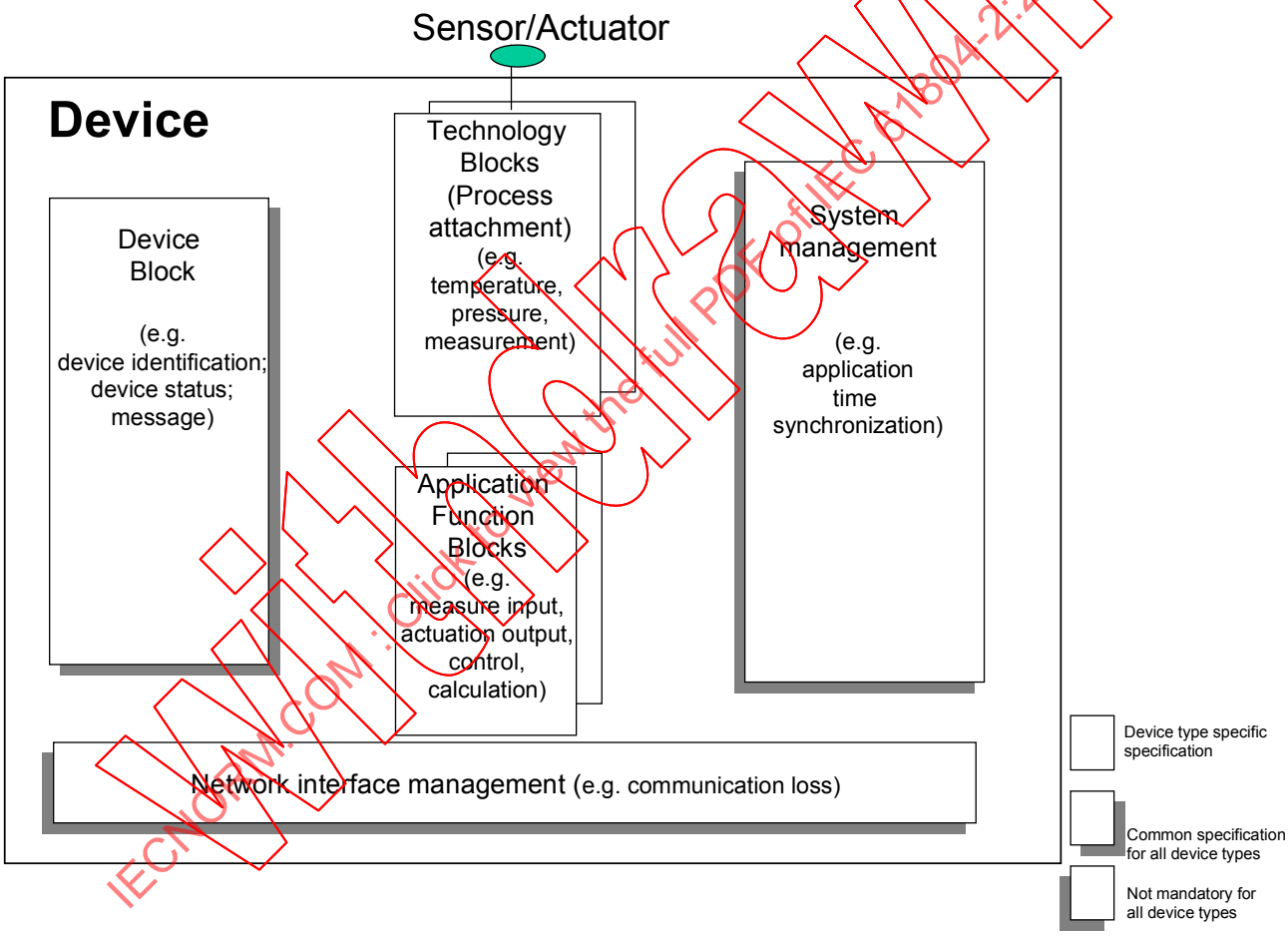


Figure 6 – Block types of IEC 61804

All devices within the scope of this standard shall have the same logical device structure, see Figure 6. The number and types of blocks, which are instantiated in a device, are device and manufacturer specific. At least, it shall have one Device Block, one application FB and one network interface management.

There is a data flow chain from signal detection through the Technology Block and FBs and vice versa. The signals between the parts of the chain are internal within the blocks or visible as linkages between blocks. The logical chain of technology and FB is called a channel. This concept is clarified in 4.2.1 and 4.2.2.

4.1.2 FB type

FBs are functional units in software, which encapsulate variables and algorithms. A FB type is defined by its behaviour. One FB contains one or more than one algorithm. The description of a FB is a list of algorithms, which are encapsulated in the FB together with the related data inputs and data outputs and parameters. There are algorithms, which are related to the process signal flow and those, which are related to other block specific algorithms. These other algorithms are called management. Parameters are related to process signal flow and management.

Graphical representation is not normative (see Figure 7). In other words, the data inputs and data outputs represents the intention of the process signal flow (conceptual definition) not the specific data that carry the according values.

The parameter table specifies all the necessary accessible data inputs, data outputs and parameters of the FB.

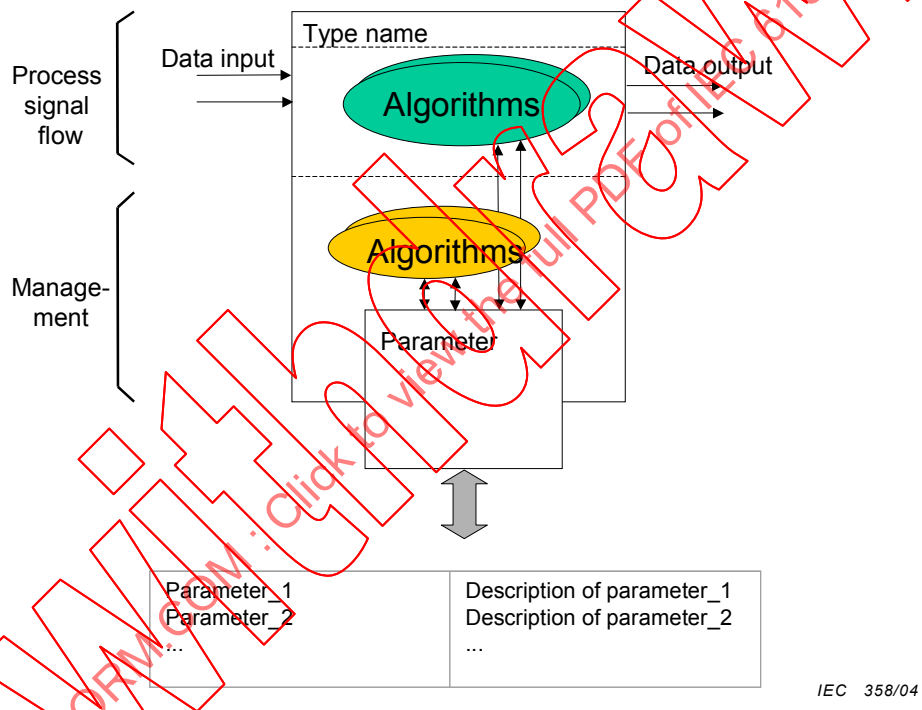


Figure 7 – IEC 61804 block overview (graphical representation not normative)

The FB is summarized by the following components:

a) data Inputs[3] which support status[4] and are related to the process signal flow only;
b) data Outputs[3] which support status[4] and are related to the process signal flow only;
c) parameters[3] and are related to the process signal flow and management;
d) maintain values to influence functions;
e) notify and make visible internal behaviour;
f) selection of functions in the signal flow;

---

[3] The decision as to which data of a FB is a data input, data output or parameter depends on specific implementations.

[4] For consistency reasons, data input/data output and status are in one structure, so that both belong to each other.

g) internal variables with memory for support of for example initialization;

h) mathematical/logical algorithm.

The influence of the FB behaviour is possible by data inputs and parameters only. The data inputs and parameters are used in the following ways:

a) data, which are used as inputs or outputs of functions (e.g. setpoint for scaling functions);

b) data, which are used as parameter of functions (e.g. limits for alarms and warnings);

c) changes of parameter data are interpreted as events which switch transitions of state automata (e.g. start, stop, resume of operation modus of devices);

d) changes of parameter data are interpreted as events, which start transactions of sequences of algorithms (e.g. start of calibration procedures).

The data name and their description have to be checked to understand the purpose of the data.

4.1.3 FB execution

Execution control of FB algorithms is a feature of each device. Different execution policies are allowed.

NOTE For example, combinations of the following execution control methods are possible and others may be added:

a) free running;

b) device internal time schedule (time synchronization), e.g. IEC 61131-3, Subclause 2.7.2;

c) device internal event triggered;

d) parameter data changes are interpreted as events (see 4.1.2);

e) system wide time synchronization (time synchronization across the communication system);

f) communication service triggered;

g) system wide event triggered (e.g. IEC/PAS 61499-1);

h) distributed execution control;

i) device internal time schedule (time synchronization).

The FB execution control within a device is only one aspect of the overall application execution control. The overall execution control is determined for example (see, for example, IEC/TR 61131-8, Subclause 3.10) by:

a) Sequence order (sequential or parallel):
   1) Execution order of blocks along the signal flow
   2) Piping of data in parallel execution
   3) Handling of loss of communication between devices
b) Synchronization:
   1) Time synchronization between device
   2) Use of time in scheduling
c) Time constraints; the following elements are covered:
   1) Block execution time
   2) Communication time delay
   3) Scan rate of measurement
   4) Actuation time
   5) Choice of block algorithms
   6) Time delay resulting of communication behaviour
d) Block execution time:
   1) Communication time delay
   2) Scan rate of measurement
   3) Actuation time
   4) Choice of block algorithms

e) Impact of exception handling:
   1) Clock error
   2) Device error
   3) Communication error.

The decision as to which technology fulfils the requirements has to be based on a detailed check of at least all these aspects. The choice of execution control method also depends on the technology level used to build the devices. So the method of FB execution control is also constrained to those available in the fieldbus used by the system.

4.1.4 Reference between IEC/PAS 61499-1, IEC/PAS 61499-2 and IEC 61804 models

The relations to IEC/PAS 61499-1 and IEC/PAS 61499-2 are given in Table 2.

Table 2 – References of model elements

IEC 61804 model element	IEC/PAS 61499-1 model element
Reference of block types	
Application FB	Application FB
Technology Block	Technology Block
Device (Resource) Block	Device (Resource) Block
Reference of FB elements	
Component Block	Component Block
Type Name	Type Name
Data Input [a]	Data Input [a]
Data Output [a]	Data Output [a]
Algorithms	Algorithms
Parameter	Parameter
Internal Variable	Internal Variable
Principle relations between EDDL elements and IEC/PAS 61499-2 transfer syntax elements [b]	
BLOCK_A, BLOCK_B	FUNCTION BLOCK
VARIABLE and CLASS INPUT	VAR_INPUT, END_VAR
VARIABLE and CLASS OUTPUT	VAR_OUTPUT, END_VAR
– [c]	ALGORITHM
VARIABLE and CLASS CONTAINED	–
VARIABLE	VAR, END_VAR

[a] The data inputs and data outputs represent the source and sink points for the process signal flow (conceptual definition) not the specific variables, which carry the according data.

[b] This is not an exact syntax reference. It is intended to show the general relations.

[c] Describing algorithms are not the intention of EDDL.

An IEC 61804 FB is an IEC/PAS 61499-1 FB without execution control and therefore has no event inputs and event outputs. The execution control of the IEC 61804 FBs algorithms are hidden (see 4.1.3).

4.1.5 UML specification of the device model

The device model definitions in 4.1.1, in Figure 5, and Figure 6 are general. To solve the ambiguity, the model is described as a UML class diagram (see Bibliography). The components are transformed to the UML language elements in Figure 8.

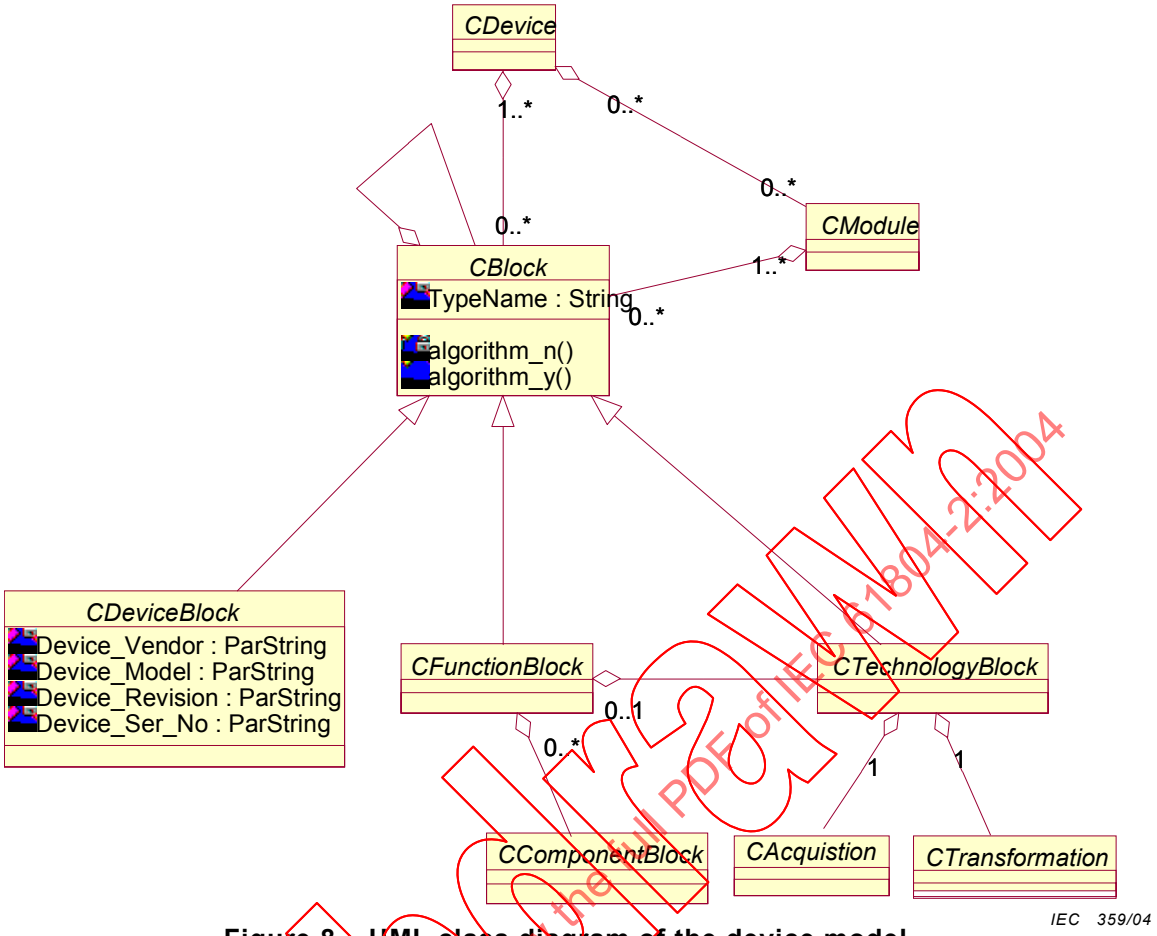


Figure 8 – UML class diagram of the device model

The following major steps are used to convert the device model into a UML class diagram:

a) the device becomes the class CDevice;

b) the module becomes the class CModule;

c) the Device Block, FB Component FB and Technology Block become CDeviceBlock, CFunctionBlock, CComponentFunctionBlock and CTechnologyBlock;

d) the block types are of the type Block which becomes CBlock;

e) a device contains a minimum of one block;

f) a device may contain modules;

g) a module contains a minimum of one block;

h) blocks can be composed out of other blocks, i.e. may be of composite FB type;

i) a block contains a minimum of zero or more parameters;

j) a block shall have algorithms which can be internal only or visible from the outside (i.e. private or public);

k) a Device Block contains the attribute Device_Vendor, Device_Model, Device_Revision and Device_Ser_No which are parameters;

l) the FB, Component FB and Technology Block contain the attribute TypeName.

NOTE The CBlock class can be referenced to the Basic FB Type Declaration of the IEC/PAS 61499-1 (see Figure C.1.4). The IEC 61804 block type has no aggregation to the ECCDeclaration class.

4.1.6 Classification of the algorithms

The following list provides common algorithms for use in application FBs, transducer blocks and device blocks.

a) Process signal Algorithms
   1) Measurement acquisition
      i) Sensor connection
      ii) Sensor range/calibration
      iii) AD conversion
      iv) Status estimation
   2) Measurement transformation
      i) Linearization
      ii) Filtering
      iii) Compensation
      iv) Scaling
   3) Measurement application
      i) Limit
      ii) Unit
      iii) Scaling
      iv) Linearization
      v) Simulation
   4) Actuation provision
      i) Amplification
      ii) Conversion
      iii) Status estimation
   5) Actuation acquisition
      See measurement acquisition for readback of actuator output value
   6) Actuation transformation
      i) Scaling
      ii) Compensation
      iii) Transition or activity limits
   7) Actuation application
      i) Limit
      ii) Unit
      iii) Scaling
      iv) Linearization
      v) Simulation

b) Management
   1) Estimation of Device Status
   2) Test
   3) Diagnosis
   4) Operating Mode

4.1.7 Algorithm description

The algorithm description is made individually for each algorithm in the appropriate language, for example plain English, Harel State Diagram or one of the IEC 61131-3 languages (for example FBD (FB diagram) or IEC 61131 ST (structured text)).

The object of the profile description is to define a general set of rules allowing identification of a device together with classification and specification of the algorithms supported by the device.

4.1.8 Input and output variables and parameter definition

For the description of the block parameters, Table 3 table shall be used. This table provides a template for describing the interface to a block. It is comparable with a data dictionary or a database.

Table 3 – Variables and parameter description template

Parameter Name	Description	Data Type	User Access Read/Write	Class m/o/c
Block class				

Parameter Name:

Identifier of the variable/parameters that are accessed within the FB. The name is valid within this specification but not normative for products on the market. The decision if a data is an input, output or parameter is application dependent.

Description:

Informative text, describing the purpose of the variable/parameter.

Data type:

The following data types are conceptual ones, i.e. they identify the signal type not the implementable data type. These will be mapped by technology profile to supported data in the following categories:

a) numeric (e.g. float, real, long real, integer);

b) enumerated;

c) boolean;

d) string (e.g. visible string, octet string);

e) array;

f) structure.

User Access Read/Write

This specifies that the variable/parameter is changeable by a remote device or not.

Class m/o/c

This specifies if the variable/parameter shall be supported within the block or not; the states are: mandatory (m), optional (o), conditional (c).

Additional parameter attributes that shall be specified when mapping IEC 61804 blocks to other FB specifications are:

a) class of recovery after power fails shall have the value N or D as follows:

   N indicates a non-volatile parameter which shall be remembered through a power cycle, but which is not under the static update code;

   D indicates a dynamic parameter that is calculated by the process, the block or read from another block.

b) default value

   indicates the value is assigned to a parameter in the initialization process for an unconfigured block.

4.1.9 Choice of variables and parameters

The block variables, parameters and algorithms included in a block will be those that are significant for the algorithm and device. As a minimum, FBs will include the variables and parameters defined in the P&ID. The names of parameters and variables are not normative.

4.1.10 Mode, Status and Diagnosis

These parameters manage and indicate channel performance. They can be reported; however, the report mechanisms are technology dependent. Reported values may also include additional items, such as time stamps, priorities, indication of possible reasons, etc.

Mode describes the operation state of a channel or FB and influences the signal flow within the channel. Examples of modes are; Manual, Automatic, Local Override, Out of Service.

Status is a characteristic aspect of a channel which may accompany information transferred within the channel i.e. FB data inputs and data outputs.

Device State describes the operational state of a device and interacts with the device technology and application blocks, it is maintained within a device by the device block.

Diagnosis is a report available from algorithms which assess channel or device internal performance. The results of these internal assessments may be used to construct generic measurement, control and actuation status information.

4.2 Block combinations

4.2.1 Measurement channel

The technology and application FBs provide a functional chain along which the process signals flow. Together they comprise a measurement channel (see Figure 9) or an actuation channel (see Figure 10).

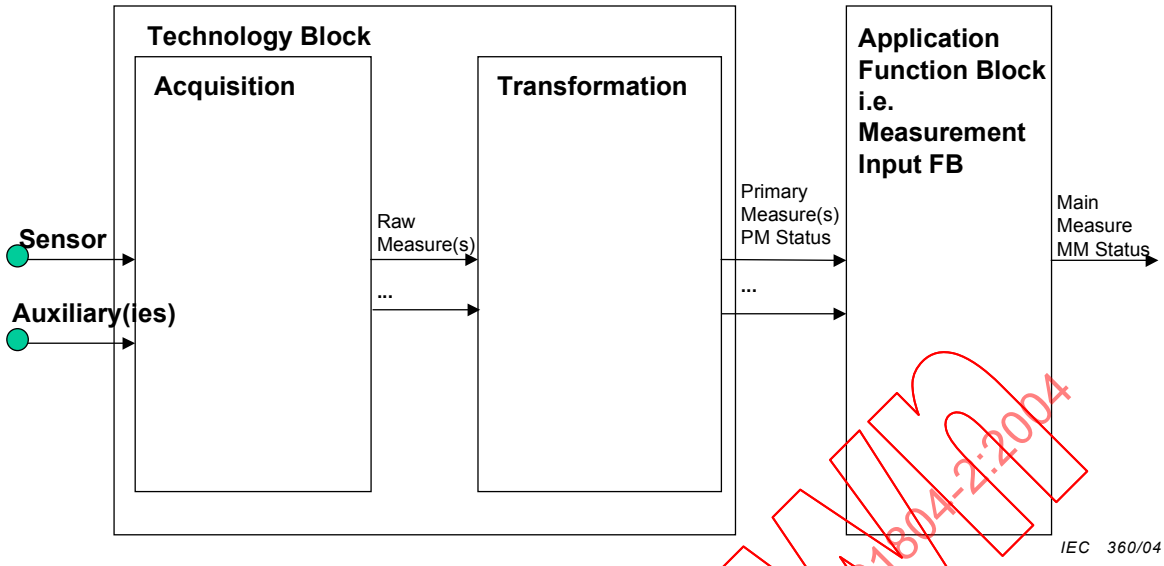


Figure 9 – Measurement process signal flow

A measurement may be accompanied by optional additional auxiliary measurements, for purposes such as compensation. The technology block provides a primary measured value and its accompanying status. Additionally, the technology block may provide other outputs – for example, diagnosis or validation information.

NOTE Additional sensor inputs may also be used and transferred by a technology block.

The application FB uses the outputs of the technology block and other internal data to generate the main measure and its accompanied status. The status is accomplished by every function in the signal flow starting with the sensor(s) until the last function in the application FB. Information from one technology block is offered to more then one application FB. A measurement channel shall consist of at least one application FB. Channels without a technology block are possible.

4.2.2 Actuation channel

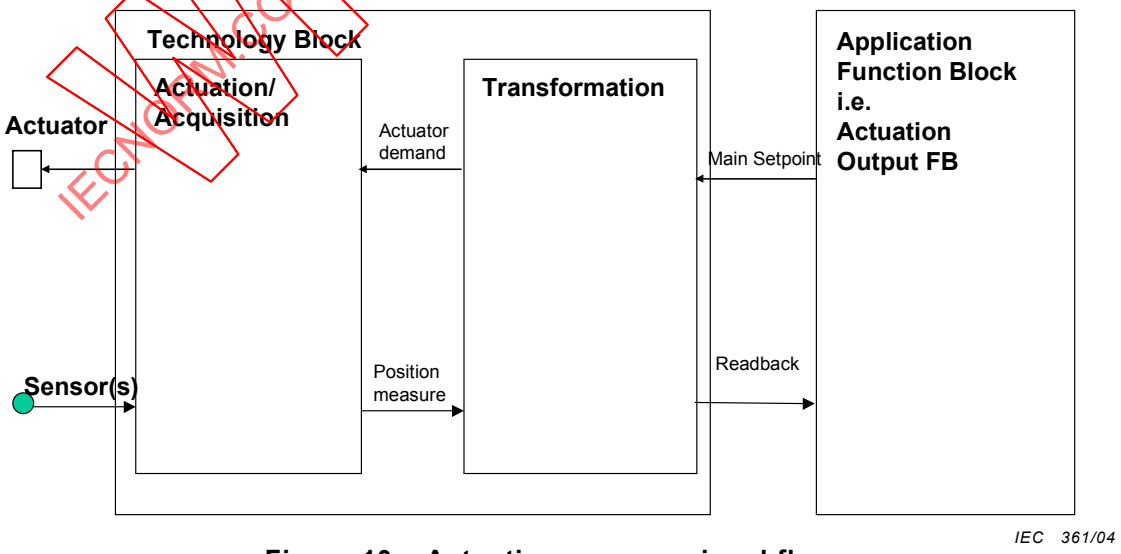


Figure 10 – Actuation process signal flow

The actuation channel is performed out of the function of the actuation signal flow and the additional measurement functions for the measurement of the current position of the actuator. If there is not a sensor for the position measurement, then the actuator demand will be used in the transformation to determine the readback value. Optionally, status values may accompany both signal flow directions and include information about the involved entities. The status accompanying the main setpoint carries information to give the technology block the opportunity to go in fail-safe position, if the main setpoint is not good. The status accompanying the readback carries information if the measure value is good or not. An actuation channel shall consist of at least one application FB. Channels without a technology block are possible.

4.2.3 Application

A complete application is supported by combinations of measurement and actuation channels together with control and calculation FBs (see Figure 11). The technology blocks are technology dependent and the other FBs are technology independent. There may be many different implementations of an application, depending on the technology used within the devices. The application may be performed by implementations using only measurement and actuation devices (i.e. complex devices able to perform measurement, control and actuation), or the application may be built from measurement and actuation devices together with controller devices and other system components.

NOTE A controller can, for instance, be integrated in the application as one calculation FB, or an actuation device can take parts of programmable functions from controller devices in terms of calculation FBs.

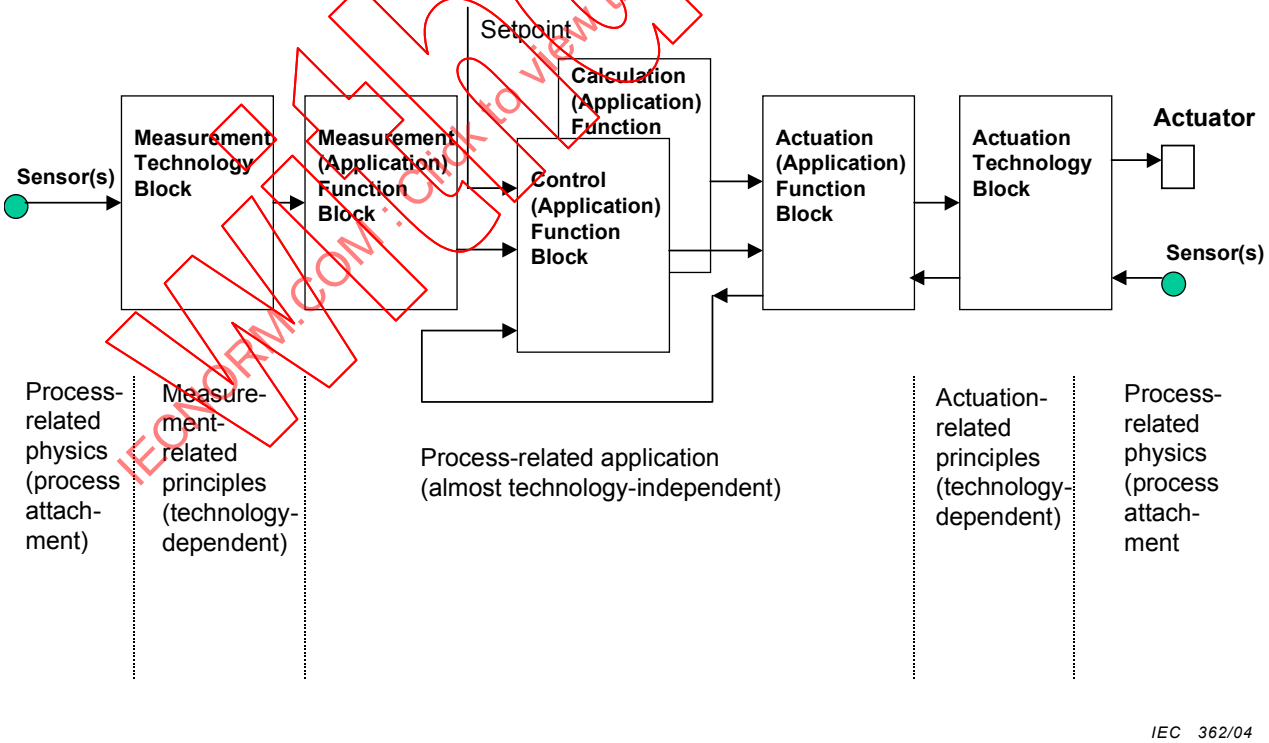


Figure 11 – Application process signal flow

4.3 EDD and EDDL model

4.3.1 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 language code like Japanese, may use a multi-byte code.

4.3.2 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.

4.3.3 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 disk 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, e.g. manufacturer, device type, revision, etc. This is used to match a specific EDD to a specific device.

4.3.4 Principles of the EDD development process

4.3.4.1 General

Creation of an EDD is a three-stage process, EDD source generation, EDD pre-processing and EDD compilation.

4.3.4.2 EDD source generation

The EDD generation process is shown in Figure 12.

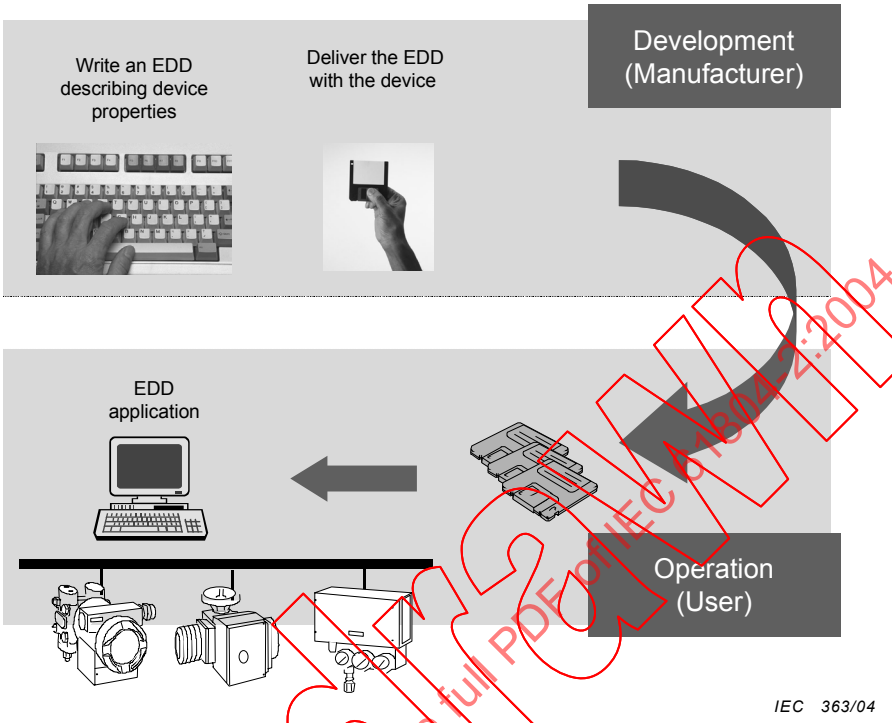


Figure 12 – EDD generation process

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.

4.3.4.3 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.

4.3.4.4 EDD compilation

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

4.3.5 Interrelations between the lexical structure and formal definitions

The lexical structure of EDDL and its elements are described in Clause 9. Formal definitions and syntax for each EDDL element are given in Clause C.6. The lexical structure and its formal description use the same name.

NOTE Instead of the specified formal definitions and syntax in Clause C.6, another specification may be developed as an additional option.

4.3.6 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 D.

4.3.7 Profiles

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

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 standard.

5 Detailed block definition

5.1 General

This selection of blocks is not intended to be complete. It is a selection of very common measurement and actuation.

5.2 Application FBs

5.2.1 Analog Input FB

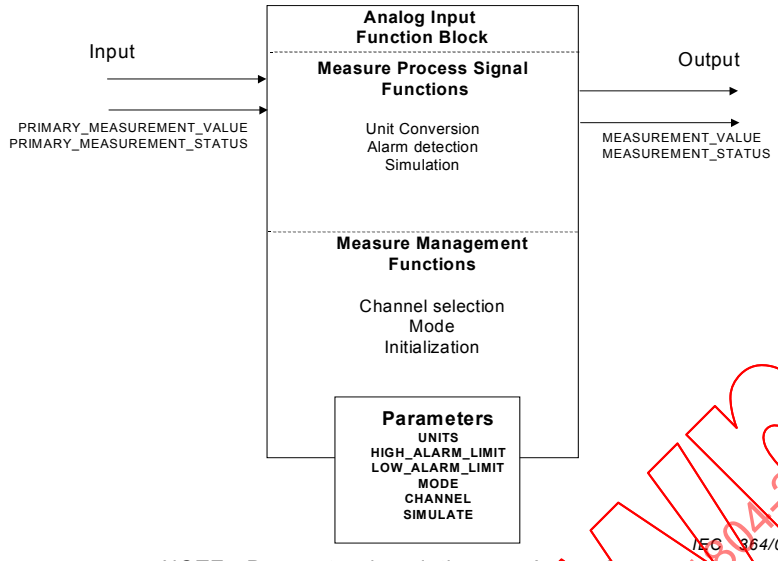
5.2.1.1 Analog Input FB Overview

The measure process signal function shall be used to convert signal(s) from a Technology Block to units appropriate for the primary measurement required for an application. The result is the MEASUREMENT_VALUE.

NOTE For example, conversion from inches of water to liters per minute. Also, this block may be used to provide operator notification that the primary measurement has detected a high or low alarm. The ability may be provided to simulate the process measurement during system checkout and testing.

Each process signal involves more information than only the value of the signal; the management parameters are generally required. Each measurement has a status, which indicates the quality of the measurement value.

The status provided by the technology block is propagated to measurement (Input) FB by the PRIMARY_MEASUREMENT_STATUS. The status is a piece of an information provided with every measurement to assist the user of measurement data (typically control functions) in assessing its utility. For example, it may be a Boolean value (valid/non-valid), a continuous value (measurement uncertainty), a discrete value, or a combination, see 5.6.1.



NOTE Parameter description see Annex A

Figure 13 – Analog Input FB

5.2.1.2 Unit Conversion

This algorithm converts the signal from a Technology block into an understandable value. That may be used directly by the operator.

The user uses the UNITS to select the engineering units in which the MEASUREMENT_ VALUE is to be displayed; for example, bar or mbar.

NOTE This algorithm may also provide information on the channel and device operating state to assist in diagnostic of management activities.

5.2.1.3 Alarm detection

The FB shall provide the optional alarm detection inside.

Examples are low alarm, high alarm, deviation, update.

When implemented, the LOW_ALARM_LIMIT and HIGH_ALARM_LIMIT values shall be compared with the MEASUREMENT_VALUE of the FB. The results are high and low alarm notification, for example, for an operator.

NOTE The way of reporting the detected alarms is technology-dependent; therefore, it is not described in this standard and shown in the relevant figure.

5.2.1.4 Simulation

This algorithm shall be used to simulate the MEASUREMENT_VALUE value to an assigned value using the SIMULATE parameter. This operation is usually carried out during commissioning, adjustment phases, or test purposes, and allows the running application to be temporarily uncoupled from the process.

5.2.1.5 Channel selection

One technology block will be used for primary final element data. Channel numbers (CHANNEL) will be defined for the measurement device when using more than one technology block.

5.2.1.6 Mode

The mode algorithm determines the source of the output for a measurement input FB based on the MODE parameter value. In Automatic mode, the measurement algorithm determines the output. When mode is set to Manual, the output of the FB is set by a different source; for example, may be set by the operator.

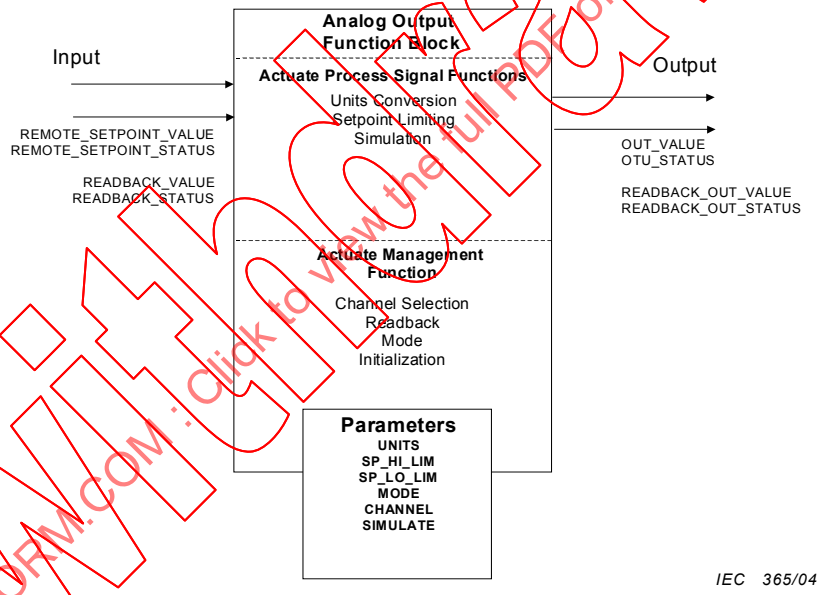
5.2.1.7 Initialization

The initialization algorithm is applied to this block and described in 5.6.3.

5.2.2 Analog Output FB

5.2.2.1 Analog Output FB overview

The actuation process signal algorithm converts REMOTE_SETPOINT_VALUE into a useful value (OUT_VALUE) for the hardware specified by the channel selection for the technology block. The feedback value (received from the actuator) is provided as the READBACK_VALUE. If the Analog Output FB is part of a cascade chain, the READBACK_OUT_ VALUE provides the actual value to the upstream FB. All these input and output parameters shall be accompanied by their status (see 5.6.1).



NOTE For parameter description, see Annex A.

Figure 14 – Analog Output FB

5.2.2.2 Unit Conversion

This algorithm converts the REMOTE_SETPOINT_VALUE to a value, which can be used by the actuator. UNITS of the REMOTE_SETPOINT_VALUE main setpoint value define the units of the setpoint. The READBACK_VALUE (i.e. the actual delivered value or the final demanded value) is also provided in the units of the setpoint.

5.2.2.3 Setpoint Limiting

The REMOTE_SETPOINT_VALUE that is provided to the FB will be limited to the setpoint lower (SP_LO_LIM) and higher (SP_HI_LIM) range limits.

5.2.2.4 Simulation

This algorithm is used to force the READBACK_VALUE and the READBACK_STATUS to assigned values through the SIMULATE parameter. The simulation can be used, for example, to simulate technology block faults. In simulation mode, the technology block ignores the Analog Actuation FB output value(s) and maintains the last value. This operation is usually carried out during commissioning, adjustment phases, or test purpose, and allows the running application to be temporarily uncoupled from the process.

5.2.2.5 Channel selection

One technology block will be used for primary final element data. Channel numbers (CHANNEL) will be defined for the Modulation Actuator Device when using more than one technology block.

5.2.2.6 Readback

This algorithm gives information about the actual delivered value of the actuator in the process.

The READBACK_STATUS information is provided to reflect the state of the actuating value. This may be a Boolean value (valid/non-valid), a continuous value (measurement uncertainty), a discrete value, or a combination.

5.2.2.7 Fail safe

The fail-safe algorithm is described in 5.6.4.

5.2.2.8 Mode

The mode algorithm determines the source of the output for the modulating actuation FB based on the MODE parameter value. In Automatic mode, the output is determined by the modulating actuation algorithm. When mode is set to Manual, the output of the FB is set by a different source; for example, it may be set by the operator.

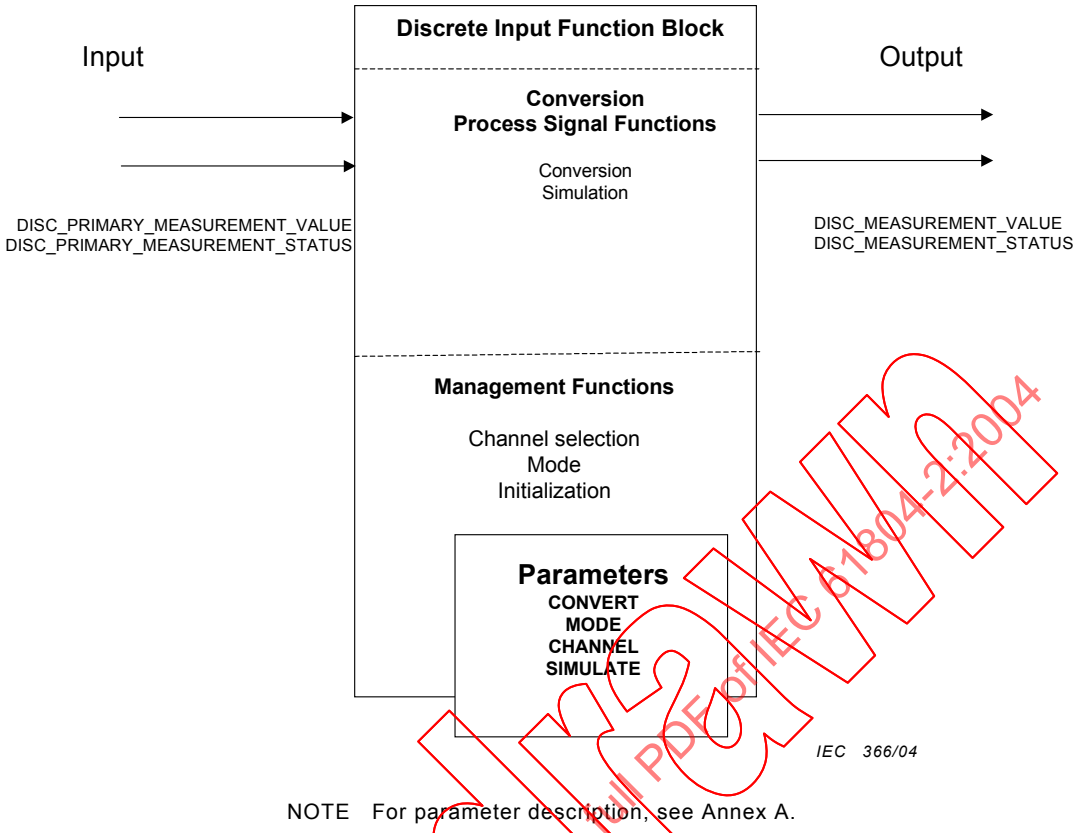
5.2.2.9 Initialization

The initialization algorithm is applied to this block and described in 5.6.3.

5.2.3 Discrete Input FB

5.2.3.1 Discrete Input overview

Discrete Inputs represent, for example, inductive, optical, capacitive, ultrasonic-, etc., proximity switches. When the digital input changes state, the discrete output changes state too.



NOTE For parameter description, see Annex A.

Figure 15 – Discrete input FB

5.2.3.2 Conversion

This algorithm converts the Boolean or discrete measure in a logical signal.

The result is the DISC_MEASUREMENT_VALUE accompanied by the DISC_MEASUREMENT_STATUS.

5.2.3.3 Channel selection

One technology block will be used for primary final element data. Channel numbers (CHANNEL) will be defined for the discrete detection device when using more than one technology block.

5.2.3.4 Simulation

This algorithm is used to force the main discrete value to an assigned value using the SIMULATE parameter. This operation is usually carried out during commissioning, adjustment phases, or test purposes, and allows the running application to be temporarily uncoupled from the process.

5.2.3.5 Mode

The mode algorithm determines source of the measure input FB output (main discrete measure) based on the MODE parameter value. In Automatic mode, the discrete measure algorithm determines the output. When mode is set to Manual, the output of the FB is set by a different source, for example, it may be set by the operator.

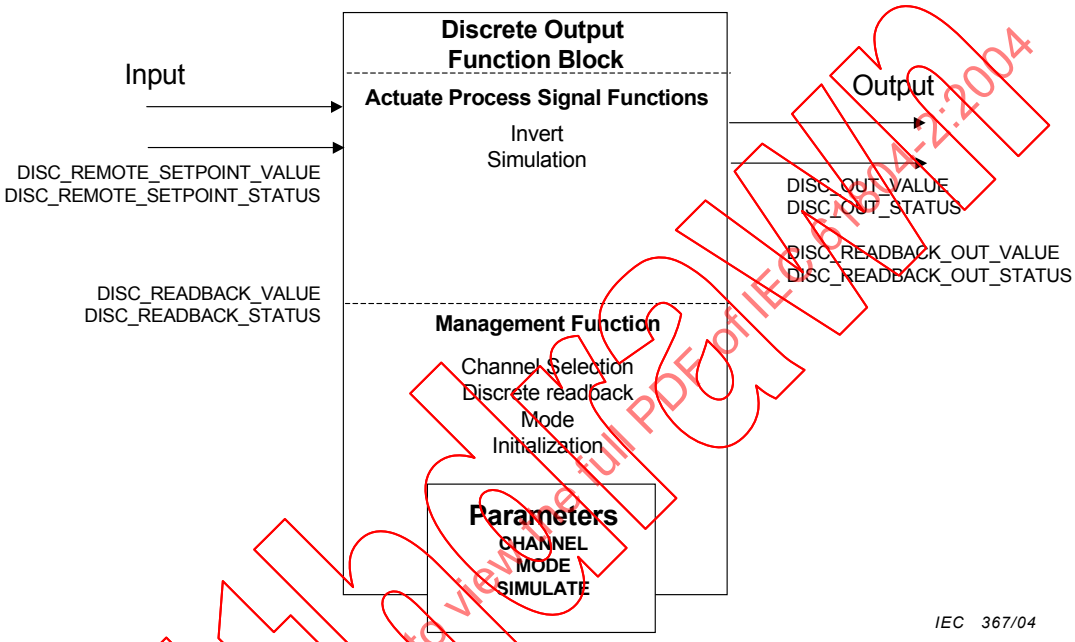
5.2.3.6 Initialization

The initialization algorithm is applied to this block and described in 5.6.3.

5.2.4 On/Off Actuation (Output) FB Discrete Output FB

5.2.4.1 On/Off Actuation (Output) FB Discrete Output FB overview

The actuation process signal algorithm converts the DISC_REMOTE_SETPOINT_VALUE value to a useful value (DISC_OUT_VALUE) for the hardware at the channel selection to the Technology block. The DISC_READBACK_VALUE defines the target value of the final element. If the Discrete Output FB is part of a cascade chain, the DISC_READBACK_ OUT_VALUE provides the actual value to the upstream FB. All these input and output parameters shall be accompanied by their status (see 5.6.1).



NOTE For parameter description, see Annex A.

Figure 16 – Discrete Output FB

5.2.4.2 Invert

Sometimes it is necessary to invert logically the DISC_REMOTE_SETPOINT_VALUE before forwarding it to the discrete actuation demand. This is done in this algorithm.

5.2.4.3 Simulation

This algorithm is used to force the DISC_READBACK_VALUE and the DISC_READBACK_ STATUS to assigned values using the SIMULATE setting. The simulation can be used for example to simulate technology block faults. In simulation mode, the technology block ignores the DISC_OUT_VALUE value and maintains the last value. This operation is usually carried out during commissioning, adjustment phases, or test purpose, and allows the running application to be temporarily uncoupled from the process.

5.2.4.4 Channel selection

One technology block will be used for primary final element data. Channel numbers (CHANNEL) will be defined for the Modulation Actuator Device when using more than one technology block.

5.2.4.5 Fail safe

The fail-safe algorithm is described in 5.6.4.

5.2.4.6 Mode

The mode algorithm determines the source of the on/off actuation FB output based on the MODE parameter value. In Automatic mode, the on/off actuation algorithm determines the output. When mode is set to Manual, the output of the FB is set by a different source, for example, it may be set by the operator.

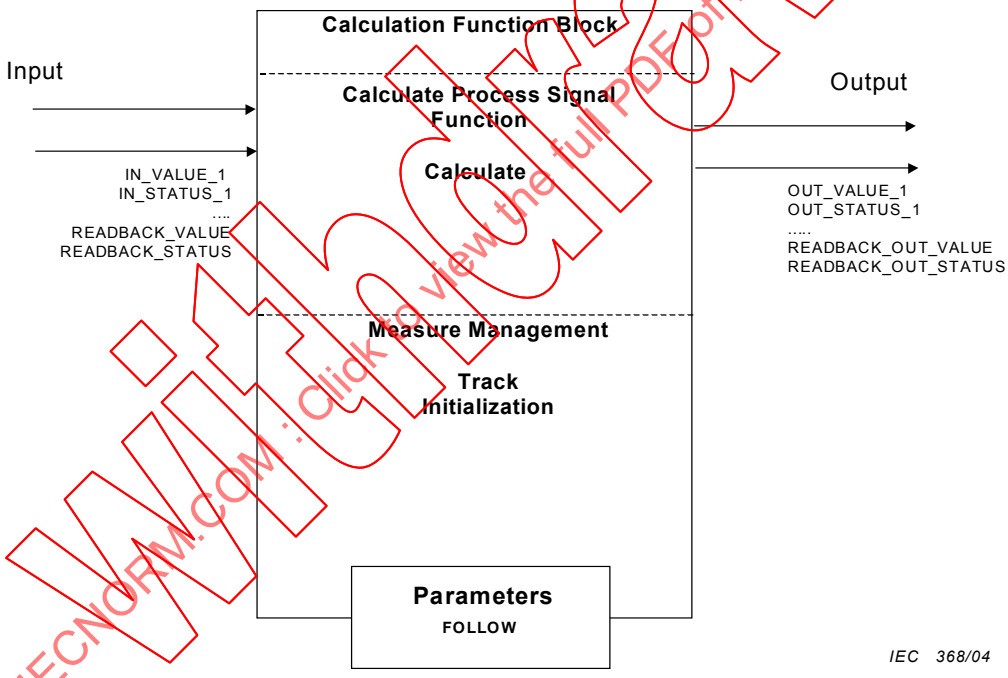
5.2.4.7 Initialization

The initialization algorithm is applied to this block and described in 5.6.3.

5.2.5 Calculation FB

5.2.5.1 Calculation FB overview

The calculation FB acts upon the input signal(s) (IN_VALUE_x) from another FB to provide an application value (OUT_VALUE_x). If the Calculation FB is part of a cascade chain, the READBACK_OUT_VALUE provides the actual value to the upstream FB and the READBACK_VALUE is provided by a downstream FB. All these input and output parameters shall be accompanied by their status (see 5.6.1).



NOTE For parameter description, see Annex A.

Figure 17 – Calculation FB

5.2.5.2 Calculate

This algorithm determines the output signal(s) based on a pre-defined algorithm and the input(s) to the FB. Examples of calculation functions are filtering, delay, input select.

5.2.5.3 Track

The track algorithm allows the FB output to be set to an input value when the FOLLOW parameter is active, i.e. non-zero in value. For example this algorithm may be used to initialize a block or to force the calculation results to a specific value.

This does not apply to all blocks.

5.2.5.4 Initialization

The initialization algorithm is applied to this block and is described in 5.6.3.

5.2.6 Control FB

5.2.6.1 Control FB overview

The control FB maintains a process input (IN_VALUE) at the setpoint value (SETPOINT) through the regulation of one or more process actuation outputs. The process input measurement is provided by an appropriate FB through the primary input connection. The primary output of the control FB regulates the process through an appropriate actuation FB. The SETPOINT defines the target value of the process measurement in mode “auto”. The Readback value and status provided by the downstream actuation block may be used in the initialization and to modify control action when the output is limited by a downstream condition.

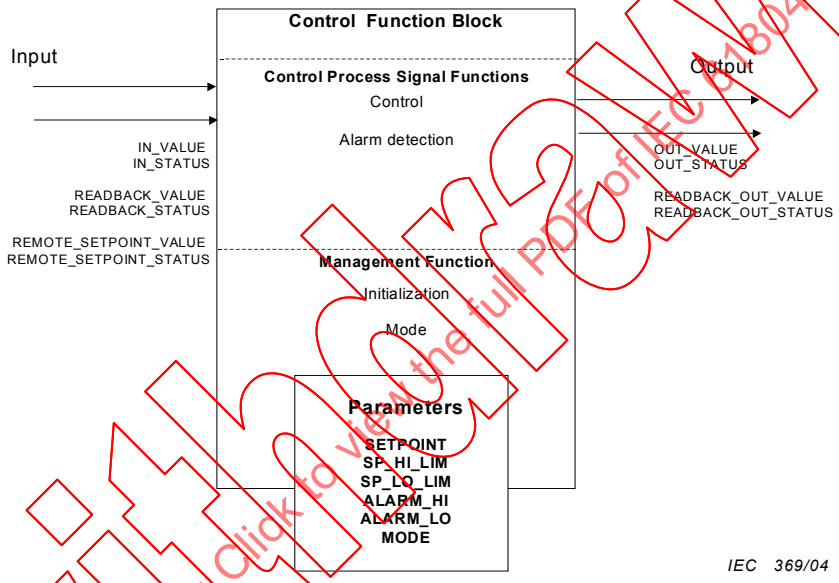


Figure 18 – Control FB

5.2.6.2 Control

This algorithm determines the FB output value that is needed to drive the primary input value to the target value specified by the SETPOINT parameter. Changes in the SETPOINT value are limited to the range specified by the SP_HI and SP_LO limits. Actions may be modified when a readback input from the downstream block indicates that a downstream condition limits the adjustment of the block output.

5.2.6.3 Alarm detection

The alarm detection is optional. When implemented, the LOW_ALARM_LIMIT and HIGH_ALARM_LIMIT values shall be compared with the primary control measurement value of the block. The results are high and low alarm notification, for example, for an operator.

NOTE The way of reporting the detected alarms is technology-dependent; therefore, it is not described in this standard and shown in the relevant figure.

5.2.6.4 MODE

The mode algorithm determines the source of the control block output based on the MODE parameter value. In Automatic mode, the output is determined by the control algorithm and the SETPOINT is specified by the operator. When mode is set to Manual, the output of the block is set by a different source; for example, it may be set by the operator.

In remote mode, the output is determined by the control algorithm and the setpoint is determined by the REMOTE_SETPOINT input from another FB.

5.2.6.5 Initialization

When the feedback status indicates that the path to the process input is blocked, the output of the FB will be set based on the readback value to provide bumpless transfer when the downstream mode is changed to remote.

5.3 Component FBs

A process control application is built out of application FBs as defined above. In addition, the application may include component FBs combined in an application-specific way and encapsulated by FBs of composite FB type. The exception handling and status handling is technology-specific and is part of the component FB definitions.

5.4 Technology Block

5.4.1 Temperature Technology Block

5.4.1.1 Temperature Technology Block overview

The algorithms of the Temperature Technology Block are summarized below.

a) Sensor connection
b) Channel range/scaling
c) AD conversion
d) Test
e) Diagnosis
f) Cold junction compensation
g) Linearization
h) Filtering
i) Initialization

The algorithms are encapsulated in the Acquisition and Transformation part of the Technology Block (see Figure 19).

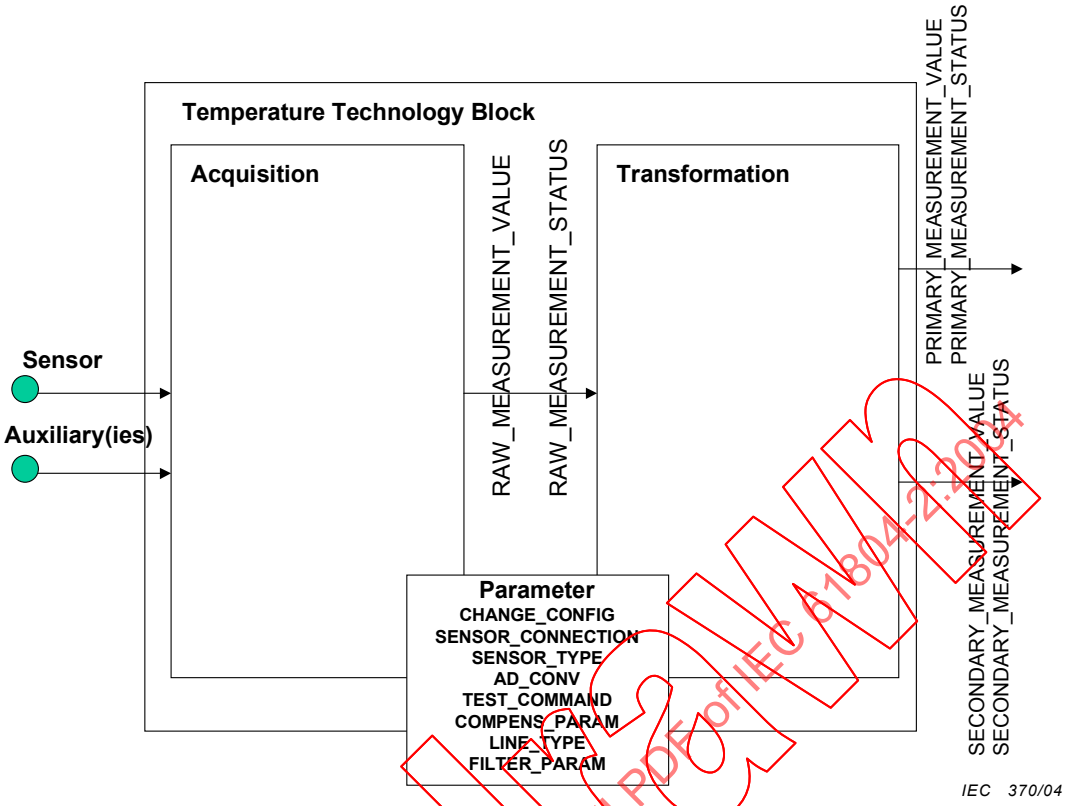


Figure 19 – Temperature Technology Block

5.4.1.2 Temperature Acquisition Functions

5.4.1.2.1 Sensor connection

The process signal is connected directly to the interface module.

There is a possibility to connect the thermo resistance with 2, 3 or 4 wires. Compensation is chosen by the parameter SENSOR_CONNECTION.

This algorithm checks the sensor link and signals a fault if there is a short-circuit or an open circuit. The wiring check is enabled/disabled via configuration (CHAN_CONFIG).

5.4.1.2.2 Channel range

This algorithm selects the sensor type which is connected to the device. According to the configuration (SENSOR_TYPE), it is necessary to differentiate between:

– electrical range (± 10 V, 0 ... 10 V, 0 ... 5 V, 1 ... 5 V, 0 ... 20 mA or 4 ... 20 mA);
– thermocouple;
– temperature probes.

Table 4 gives an example of several types of sensor.

Table 4 – Example of temperature sensors of Sensor_Type

Symbol	Description
Type B	Platinum - 30% Rhodium/ Platinum - 6% Rhodium
Type C	Tungsten - 5% Rhenium/Tungsten - 26% Rhenium
Type D	Tungsten - 3% Rhenium/Tungsten - 25% Rhenium
Type E	Chromel/Constantan
Type G	Tungsten/Tungsten - 26% Rhenium
Type J	Iron/Constantan
Type K	Chromel/Alumel
Type L	Platinel 5355/Platinel 7674
Type N	Nicrosil/Nisil
Type R	Platinum 13 % Rhodium/ Platinum
Type S	Platinum 10 % Rhodium/ Platinum
Type T	Copper/Constantan
Pt50	Platinum 50 Ω
Pt100	Platinum 100 Ω
Pt200	Platinum 200 Ω
Pt500	Platinum 500 Ω
Pt1000	Platinum 1 000 Ω
Ni10	Nickel 10 Ω
Ni50	Nickel 50 Ω
Ni100	Nickel 100 Ω
Ni120	Nickel 120 Ω
Cu10	Copper 10 Ω
Cu25	Copper 25 Ω
Cu100	Copper 100 Ω
NOTE The temperature range can be the default range of the selected thermocouple or temperature probe defined in tenths of degree (e.g. - 600 to + 11 000 tenths of °C for a Ni 1 000 probe).	

5.4.1.2.3 AD Conversion

Digitalization of input measurement analogue signal, according to the parameter set during configuration (ADCONV).

5.4.1.2.4 Test

Many test strategies are possible, for example, switching the input from the sensor to a reference signal and checking the output of the technology block against the expected value, in order to assess correct operation.

Test results then contribute to the status information processing. During tests it is recommended that the output of the connected AB maintains the previous value or other 'best estimate' of the true current value.

This algorithm is started by the TEST_COMMAND parameter, which is optional, and its implementation is manufacturer-specific.

5.4.1.2.5 Diagnosis

This algorithm is device-specific to assess internal performance of the related channel. The results of internal assessments are used to construct the generic measurement status information. Technology-specific report mechanisms provide according status information, for example, to maintenance planning.

5.4.1.3 Initialization

The initialization algorithm is applied to this block and described in 5.6.3.

5.4.1.4 Temperature Transformation Functions

5.4.1.4.1 Cold-junction Compensation

The voltage generated from thermocouple is compensated with a reference junction value. COMPENS_PARAM definse the type of compensation. The type of cold junction compensation is either Internal or External (Internal: the device itself measures the reference junction temperature via an internal mounted sensor).

5.4.1.4.2 Linearization

Thermocouple and RTD values are laniaries and compensated internally. The linearization is done according to the IEC 60584-1 reference standard for the thermocouple curve. Optionally, the manufacturer may offer an additional user-defined linearization. LINE_TYPE defines the linearization curve coefficients.

5.4.1.4.3 Filtering

A filtering is performed on the measure laniaries and compensated.

With the FILTER_PARAM, the filter efficiency shall be selected, for example 1 s, 2 s, 5 s, etc.

5.4.1.4.4 Initialization

The initialization algorithm is applied to this block and described in 5.6.3.

5.4.2 Pressure Technology Block

5.4.2.1 Pressure Technology Block overview

The algorithms of the Pressure Technology Block are summarized below.

a) Sensor connection
b) Channel range/Scaling
c) Sensor calibration
d) Test
e) Diagnosis
f) Linearization
g) Filtering
h) Temperature compensation
i) Initialization

The algorithms are encapsulated in the Acquisition and Transformation part of the Technology Block (see Figure 20).

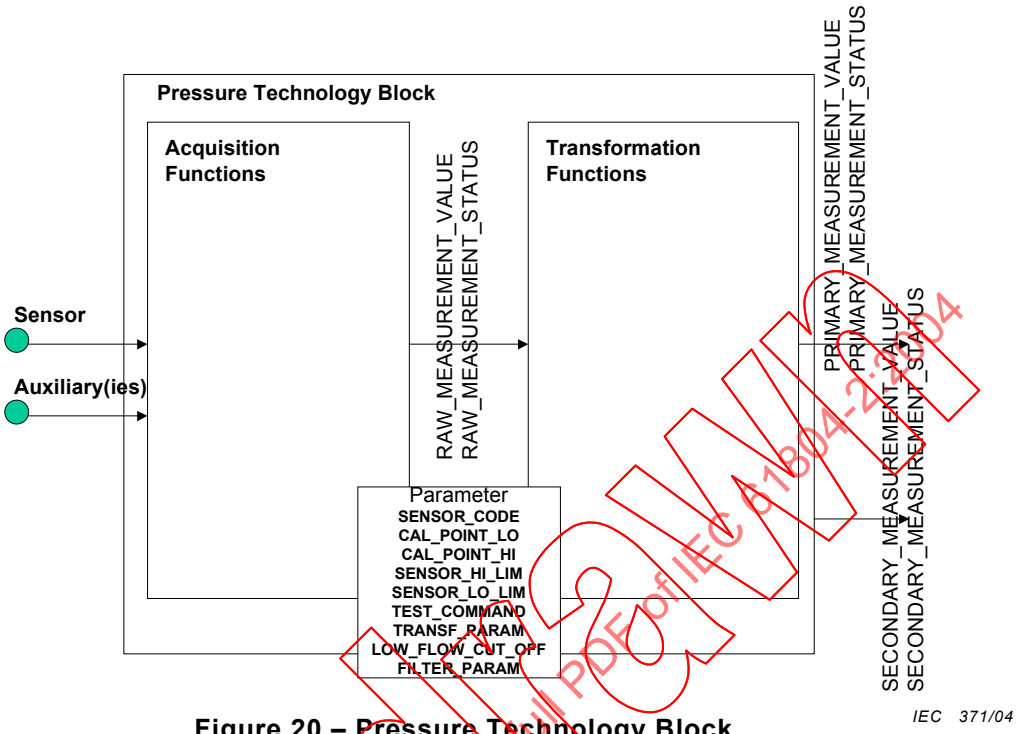


Figure 20 – Pressure Technology Block

5.4.2.2 Pressure Acquisition Functions

5.4.2.2.1 Sensor connection

There is a possibility to connect the different pressure or differential pressure sensors to the transmitter. Compensation is chosen by the parameter SENSOR_CODE depending from the measurement principle.

5.4.2.2.2 Channel range scaling

This algorithm selects the display format in which the measurements are supplied to the user. The SENSOR_HI_LIM and SENSOR_LO_LIM parameter define the maximum and minimum values the sensor is capable of indicating.

5.4.2.2.3 Sensor calibration

The calibration process is used to match the channel value combined with the applied input. The calibration of the sensor itself is not changed, because that is a factory procedure. Four parameters are defined to configure this process: CAL_POINT_HI, CAL_POINT_LO, SENSOR_HI_LIM and SENSOR_LO_LIM. The CAL_* parameters define the highest and lowest calibrated values for this sensor.

5.4.2.2.4 Test

Many test strategies are possible, for example, switching the input from the sensor to a reference signal and checking the output of the technology block against the expected value, in order to assess correct operation.

Test results then contribute to the status information processing. During tests it is recommended that the output of the Application Block maintains the previous value or other 'best estimate' of the true current value.

This algorithm is started by the TEST_COMMAND parameter which is optional, and its implementation is manufacturer-specific.

5.4.2.2.5 Diagnosis

This algorithm is device specific to assess internal performance of the related channel. The results of internal assessments are used to construct the generic measurement status information. Technology specific report mechanisms provide according status information for example to maintenance planning.

5.4.2.2.6 Initialization

The initialization algorithm is applied to this block and described in 5.6.3.

5.4.2.3 Pressure Transformation Functions

5.4.2.3.1 Linearization

Pressure sensor values are laniaries and compensated internally. Generally, a linearization is realized in the factory to meet initial accuracy. Additional linearization is done by using the TRANSF_PARAM parameter if flow or level measurement is applied with the pressure transmitter. The square root function is chosen as well as user-defined linearization tables. LOW_FLOW_CUT_OFF parameter determines the starting point for flow measurement at the lowest level.

5.4.2.3.2 Filtering

Filter values are selected (no filter, low level of filtering, medium level of filtering, high level of filtering) in the according FILTER_PARAM. The filtering is done on the measure that is linearized and compensated.

5.4.2.3.3 Temperature compensation

Usually the pressure of a liquid or gas is dependent on its temperature. The measured pressure value is compensated with the according temperature using this algorithm.

5.4.2.3.4 Initialization

The initialization algorithm is applied to this block and described in 5.6.3.

5.4.3 Modulating Actuation Technology Block

5.4.3.1 Modulating Actuation Technology Block overview

The elementary algorithms and parameters of modulated actuation are summarized below. Valves as well as motor frives are represented because the functions do not include technology details:

a) amplification;

b) readback measurement;

c) output limits;

d) self-calibration;

e) failsafe;

f) diagnosis;

g) test;

h) initialization.

A graphical representation with the inputs (left), the outputs (right) and the parameters (bottom) is used.

NOTE The inputs and outputs are logical connections and they do not always represent the signal flow from the view of the automation application (process control) or the process itself.

The algorithms are encapsulated in the Acquisition and Transformation part of the Technology Block (see Figure 21).

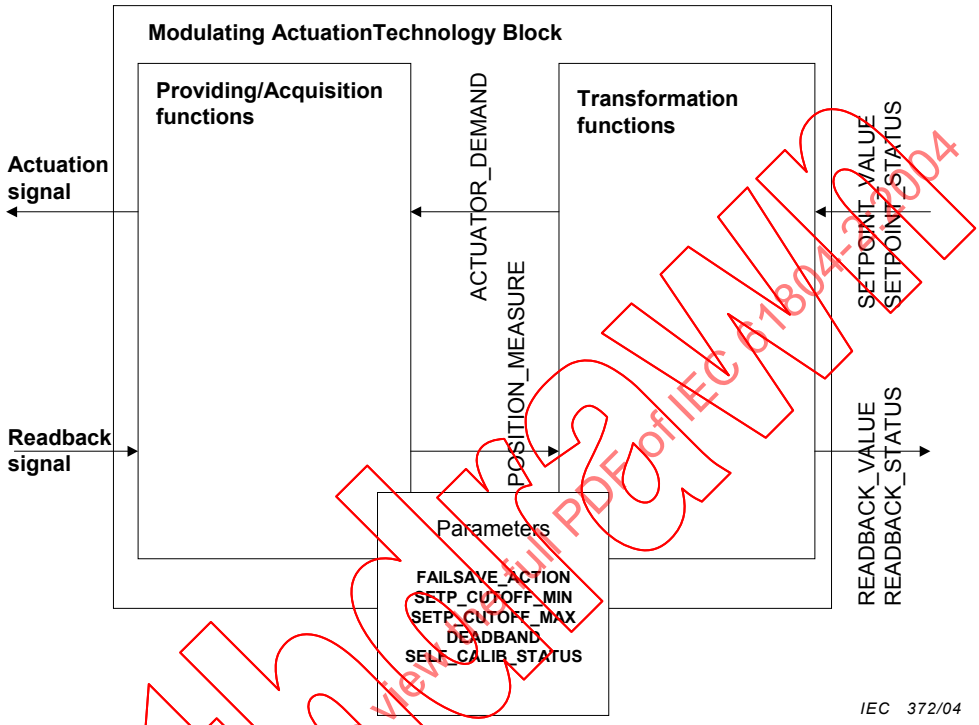


Figure 21 – Modulating actuation technology block

5.4.3.2 Providing and Acquisition Functions

5.4.3.2.1 Amplification

The block provides as an output a signal (actuation signal) from the ACTUATOR_DEMAND for the final element (e.g. a valve or motor). The final element modifies the process in response to this actuation demand output sent from the AB to the technology block (SETPOINT_VALUE).

5.4.3.2.2 Readback measurement

The block measures the actual readback signal from the final element and converts it to the transfer part of the technology block (POSITION_MEASURE).

5.4.3.2.3 Fail safe

The fail-safe algorithm is described in 5.6.4.

5.4.3.2.4 Test

Many test strategies are possible; for example, driving the actuator in a defined range and check the measured values, in order to assess correct operation. Test results then contribute to the status information processing. During tests it is recommended that the output of the test reflects the actual actions.

This algorithm is started by the TEST_COMMAND parameter, which is optional, and its implementation is manufacturer-specific.

5.4.3.2.5 Diagnosis

This algorithm is device-specific to assess the internal performance of the related channel. The results of internal assessments are used to construct the generic measurement status information. Technology-specific report mechanisms provide additional status information for example to maintenance planning.

5.4.3.2.6 Initialization

The initialization algorithm is applied to this block and described in 5.6.3.

5.4.3.3 Transformation Functions

5.4.3.3.1 Output limits

When the setpoint goes below the defined SETP_CUTOFF_MIN limit, the output (actuation signal) goes to the minimum value. With an electro-pneumatic actuator, this is done by venting/filling the actuator. With a variable-speed actuator, the actuator goes to the stopped condition.

When the setpoint goes above the defined SETP_CUTOFF_MAX limit, the output (actuation signal) goes to the maximum value. With an electro-pneumatic actuator, this is done by totally ventilation/filling of the actuator. With a variable speed actuator, the actuator goes to the full value condition.

5.4.3.3.2 Self-calibration

The procedure of self-calibrating is manufacturer-specific. The following status information (SELF_CALIB_STATUS) are typical examples:

Undetermined, Aborted, Error in mechanical system, Timeout, Aborted by means of Emergency override, Zero point error, Success.

5.4.3.3.3 Deadband

There is a deadband in which the changes of SETPOINT_VALUE does not affect the actuation signal. This is indicated in the DEADBAND parameter.

5.4.3.3.4 Initialization

The initialization algorithm is applied to this block and described in 5.6.3.

5.4.4 On/Off Actuation Technology Block

5.4.4.1 On/Off Actuation Technology Block overview

The elementary algorithms and parameters of the device are summarized below.

a) Signal conversion
b) Signal detection
c) Self-calibrating
d) Count limits
e) Failsafe
f) Test
g) Diagnosis
h) Initialization

A graphical representation with the inputs (left), the outputs (right) and the parameters (bottom) is used.

NOTE Both simple and complex implementations are available using various technologies.

5.4.4.2 Technology/Providing and Acquisition Functions

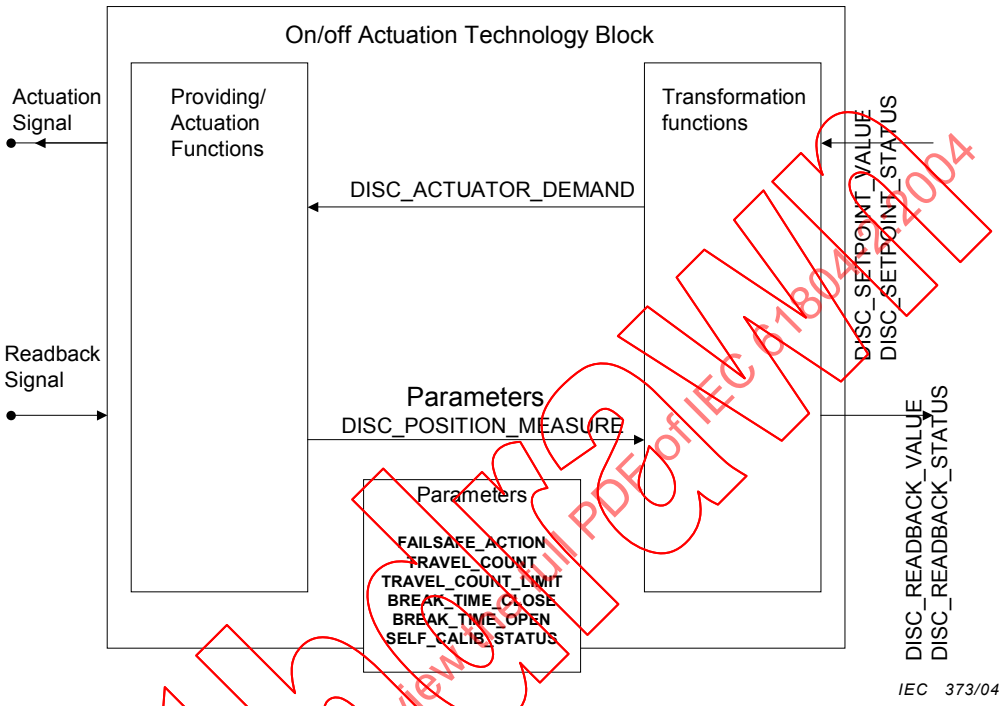


Figure 22 – On/Off Actuation Technology Block

5.4.4.2.1 Signal conversion

The block provides as an output a two-state Signal (Discrete actuation signal) from the DISC_ACTUATOR_DEMAND to the final element (e.g. a relay or valve). The final element modifies the process in response to this discrete actuation demand output sent from the application block to the technology block (DISC_SETPOINT_VALUE).

5.4.4.2.2 Signal detection

The block receives the actual demanded state (e.g. the discrete position signal) from the final element and converts it to the transfer part of the technology block (DISC_POSITION_MEASURE).

5.4.4.2.3 Break time

The actuator needs a certain period of time to switch. The break time function in the technology block provides an adjustable dead time between a new DISC_SETPOINT_VALUE value and the change of the DISC_ACTUATOR_DEMAND by the parameters BREAK_TIME_CLOSE and BREAK_TIME_OPEN.

5.4.4.2.4 Fail safe

The fail-safe algorithm is described in 5.6.4.

5.4.4.2.5 Test

Many test strategies are possible; for example, switching the actuator on and off and checking actual reached positions, in order to assess correct operation. Test results then contribute to the status information processing. During tests it is recommended that the output of the AB reflect the actual actions.

This algorithm is started by the TEST_COMMAND parameter, which is optional, and its implementation is manufacturer-specific.

5.4.4.2.6 Diagnosis

This algorithm is device specific to assess the internal performance of the related channel. The results of internal assessments are used to construct the generic measurement status information. Technology-specific report mechanisms provide relevant status information for example to maintenance planning.

5.4.4.2.7 Initialization

The initialization algorithm is applied to this block and described in 5.6.3.

5.4.4.3 Transformation Functions

5.4.4.3.1 Count limits

This algorithm counts the numbers of cycles (TRAVEL_COUNT) of an actuator. A cycle is two successive transitions from one state to the other and back to the first. The detection of transitions and the count function is manufacturer-specific. The count is often used internally to assist diagnosis and the TRAVEL_COUNT_LIMIT can trigger a suitable maintenance report.

5.4.4.3.2 Self-calibrating

The procedure of self-calibrating is manufacturer-specific. The following status information (SELF_CALIB_STATUS) are recommended:

Undetermined, Aborted, Error in mechanical system, Timeout, Aborted by means of Emergency override, Zero point error, Success.

5.4.4.3.3 Initialization

The initialization algorithm is applied to this block and described in 5.6.3.

5.5 Device (Resource) Block

5.5.1 Identification

The Device provides documentation information electronically to assist the user of a device (in particular a Control operator/algorithm) in checking the device type and revision. For the different phases of the device life cycle (Design, commissioning, documentation (on-line), it is absolutely necessary to have an unambiguous identification of the devices. Therefore, the following parameters are supported:

- DEVICE_VENDOR
- DEVICE_MODEL
- DEVICE_REVISION
- DEVICE_SER_NO for identification of multiple devices of the same type is optional.

5.5.2 Device state

The Device synthesized status is devoted to assist the user of a device (in particular a Control operator/algorithm) in assessing its remaining capabilities and to adapt accordingly its strategies. This status is called DEVICE_STATUS.

As an example the following state models are provided to aid in understanding the relevant device behaviour. Behaviour is described using a state table, a Harel state model and a transition table.

Table 5 – Device status state table

State	Description
NETWORK EXECUTING	Initial state of the device. Device is capable of responding to network commands for normal operation. The processor is running
NETWORK FAULTED	The normal operation of the device is not available, because the device functionality is not accessible through the network
APPLICATION EXECUTING	Initial state of the application. Device is available for operation (normal, test and fault detection)
NORMAL	The device is available for normal operation including the reporting of detected diagnostics and process alarms
AUTOMATIC	The device processes the value from the transmitter according to all algorithms (Scaling, filtering, limit checks, engineering unit conversion)
MANUAL	This state is used to force the main measurement to an assigned value
LEARNING	The device is performing an automatic adjustment of some parameters (e.g. Functional Threshold). This state is optional, it depends on the device
FAULTED	The device is not available for normal operation. Within this state, additional sub categories of fault status may be reported. Examples are: diagnosis, event time stamp and maintenance priority
TEST	Device is performing test. This state is optional, it depends on the device

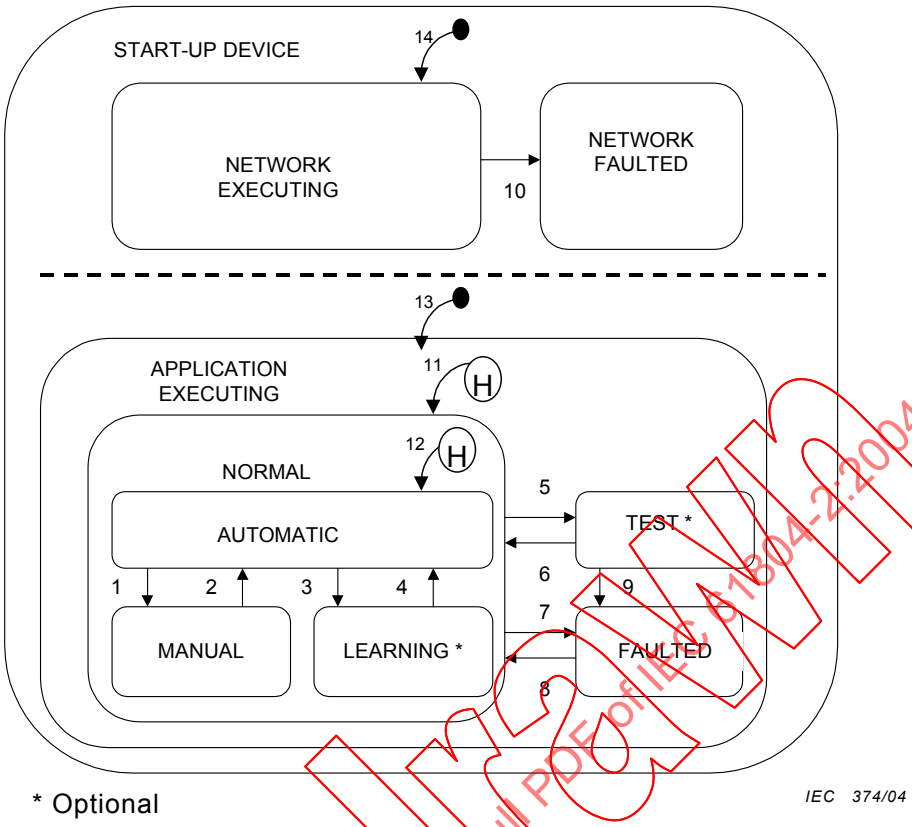


Figure 23 – Harel state chart

Table 6 – Device status transition table

Transition	From state	To State	Description
1	AUTOMATIC	MANUAL	A control command with the operating mode "Manual" is received by the device
2	MANUAL	AUTOMATIC	A control command with the operating mode "Automatic" is received by the device
3	AUTOMATIC	LEARNING	Not mandatory for all devices
4	LEARNING	AUTOMATIC	Not mandatory for all devices
5	NORMAL	TEST	Not mandatory for all devices
6	TEST	NORMAL	Not mandatory for all devices
7	NORMAL	FAULTED	A fault is detected
8	FAULTED	NORMAL	Fault reset
9	TEST	FAULTED	Not mandatory for all device
10	NETWORK EXECUTING	NETWORK FAULTED	Communication port failed, processor failed
11	APPLICATION EXECUTION	NORMAL or state before restart	Initialization of application to provide diagnosis and alarm information
12	NORMAL	AUTOMATIC or state before restart	Application run now in AUTOMATIC or states LEARNING or MANUAL and recovering the states according the device data
13	Power off	APPLICATION EXECUTION	Initialization to device application
14	Power off	NETWORK EXECUTING	Initialization of the communication

5.5.3 Message

The device provides memory space to store user information arising during the lifetime of the device. The user, for example, service staff or maintenance operator, writes textual information in this parameter. For example, it can be used for documentation purposes.

5.5.4 Initialization

The initialization algorithm is applied to this block and described in 5.6.3.

5.6 Algorithms common to all blocks

5.6.1 Data Input/Data Output status

The synthesized output status is determined by a block based on its algorithm execution results, which for example consider block inputs, block parameters, Diagnosis and Device State. It is provided to assist the user of a device or a measurement (in particular a Control operator/algorithm) when assessing its current performance capabilities and to adapt accordingly its strategies.

For example, input status is used by some FBs to change MODE and execute alternative algorithms.

5.6.2 Validity

Each FB can optionally offer a validity function, which provides a more detailed information about the quality of the measurement than one expressed with the input/output status. In this case the FB has to record in the parameter list the relevant contained parameters (e.g. Uncertainty_Value and Uncertainty_Status). Validity functions have to be separated from status functions.

NOTE The distribution of the validity information can be done in an acyclic or in a cyclic way.

5.6.3 Restart Initialization

Many process control applications require control strategies to take pre-defined initialization actions when restarting components and devices in the process control system. This capability is commonly called a Restart Initialization function. The restart initialization actions are highly dependent on the control system technologies and are often configured uniquely for the particular process application.

The following optional behaviors may apply:

– first activation of a new device;
– cold restart of a device (extended power failure);
– warm restart of a device (short power failure);
– return of a device from fail-safe.

NOTE 1 This may be implemented as part of device management, FB management, mode or application program.

For example, output technology blocks include defined default values for input (channel) parameters and the associated block functions to drive the output hardware to its un-powered state when the technology block input (channel) is not configured (i.e. the technology block input (channel) is not connected to a FB output).

NOTE 2 The physical device is represented by the Device Block. The initialization of the device block is the visible initialization of the physical device.

5.6.4 Fail-safe

In many process control applications, it is critical for control strategies and devices to take safe pre-defined actions in the event of a failure of strategies, components, or devices in the process control system. This capability is commonly called a fail-safe function. The following optional behaviours may apply:

- a resource fail-safe command, when set, will cause appropriate technology and FBs within the resource to execute their defined fail-safe actions;
- also, a resource fail-safe disable command, when set, will disable all fail-safe actions within the resource;
- initiate a fail-safe command on detection of lack of communication with other devices or resources within the system.

The particular pre-defined actions taken are highly dependent on the process application. The precise implementations of fail-safe functions are highly dependent on the control system technologies.

For example, resource blocks in some technologies and applications include parameters and functions to provide fail-safe action of device hardware and blocks associated with the resource. Fail-safe disable is enabled with a hardware jumper in this example profile. When fail-safe disable is active, the resource sends fail-safe disabled notifications to other appropriate resources in the system.

For example, technology blocks in some technologies and applications include parameters and functions to provide fail-safe action of device hardware associated with the block. For example, a technology block in one profile will execute pre-defined fail-safe actions on detection of bad channel or hardware values. The technology block will also execute pre-defined fail-safe actions on receipt of a resource block fail-safe command.

For example, control, calculation, and output FBs in some technologies and applications include parameters and functions to provide fail-safe action of control functionality within the block. For example, a FB in one technology profile will execute pre-defined fail-safe actions on detection of bad input, output, or transfer values. The FB will also execute pre-defined fail-safe actions on receipt of a resource fail-safe command. When fail-safe is active, these FBs send fail-safe notifications to appropriate resources in the system via their own resource.

5.6.5 Remote Cascade Initialization

A control block Out value may connect to the Remote Setpoint of an output or control block. The downstream block will be set its setpoint to this remote setpoint input value when the block mode parameter is set to remote cascade. To prevent the block setpoint from changing when the mode transitions from auto or manual to remote cascade, the output of the block providing the remote setpoint value must match the setpoint. To allow this co-ordination, the Readback Out value and status of the downstream block are connected to the Readback value and status of the upper block. The Readback Out value must reflect the block Setpoint or In_Value value. The Readback Out status reflects the mode and initialization state. Similarly, the control block OUT Status should reflect action taken by the block based on its Readback input.

When the control block Readback status indicates that the downstream block is not in Cascade mode, then its Out Value will be set to the Readback value. When the mode of the downstream block transitions to Remote Cascade, then its Readback status should indicate that initialization is required. Only after the control block has taken action on this initialization request should its Out status indicate that initialization is complete. Once the Remote Input status reflects that initialization is complete, then the block must set the Setpoint to the Remote Setpoint value and provide a status indicating normal operation in its Readback Out status.

6 FB Environment

The FB Environment is composed of additional object and block types to the types defined in 4.1.1. These object and blocks are

- Link Block;
- Alert Block;
- Trend Block.

NOTE The FB Environment is very platform and technology dependent.

7 Mapping to System Management

The mapping to System Management is an open issue regarding IEC 61158 series. Therefore it is not done within this specification.

NOTE Fieldbus specific solutions may define their own mapping without changing the definition of this standard.

8 Mapping to Communication

To provide a systematic mapping to communication networks the ISO OSI Reference Model has to be used. Regarding the application representation the model shown in Figure 24 is used.

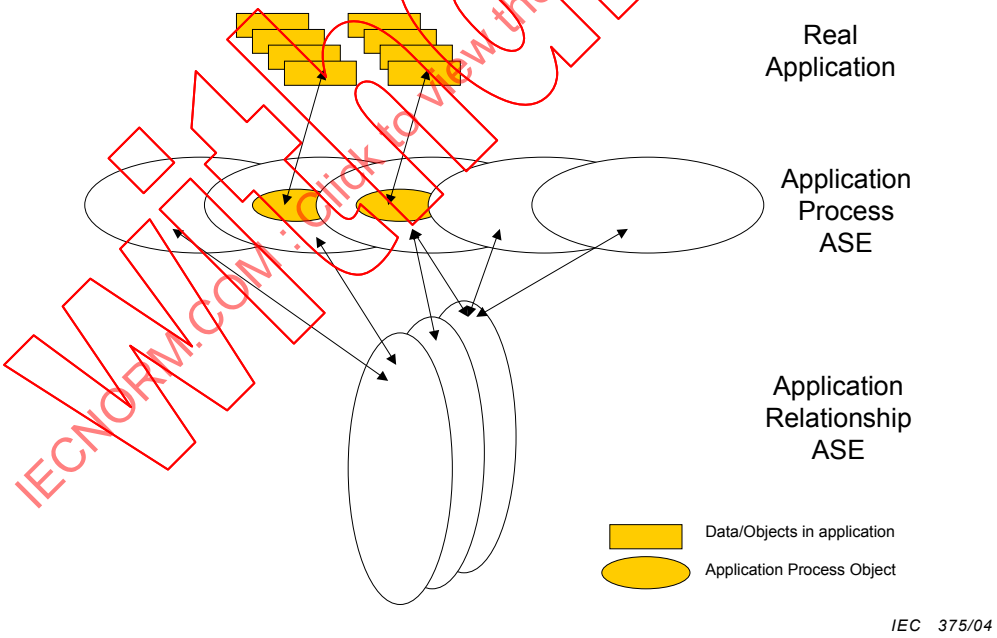


Figure 24 – Application structure of ISO OSI Reference Model

The real application data inputs, data outputs and parameters and object are represented by so-called Application Process Objects (APOs) which are managed by so-called Application Process Application Service Entities (AP ASEs) (see OSI Reference Model). These AP ASEs communicate via so-called Application Relationship ASEs.

For example, a client server relationship is modelled as shown in Figure 25.

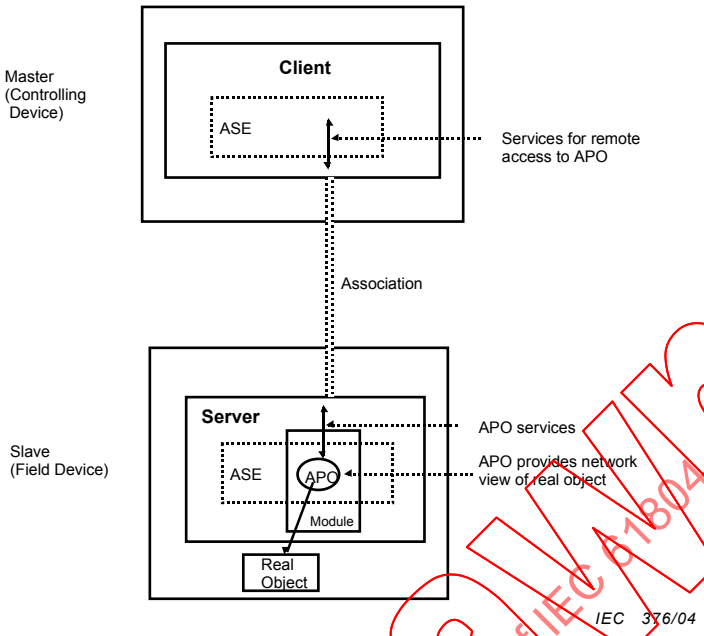


Figure 25 – Client/Server relationship in terms of OSI Reference Model

IEC 61158 uses exactly this model. Therefore, mapping has to use the same.

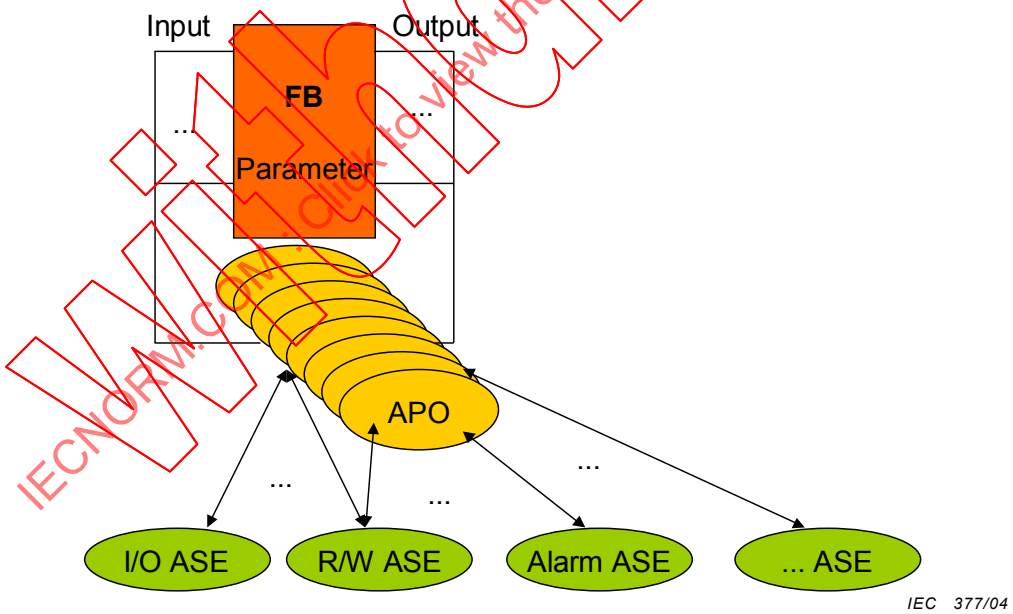


Figure 26 – Mapping of IEC 61804 FBs to APOs

The proposed mapping rules are as follows:

- Inputs, Outputs, parameter and the blocks themselves should be mapped to according APOs.
- For each APO the allowed ASEs have to be defined. More than one ASE per APO is possible.

The mapping to a fieldbus according IEC 61158 or any other communication system has to be done by the appropriate expert group of the communication system.

9 Electronic Device Description Language

9.1 Overview

9.1.1 EDDL features

The EDDL is a structured and interpretative language for describing device properties. Also the interactions between the devices and the EDD runtime 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 (e.g. choice of constructs, number of recursions, selection of options), Profiles supported by some industrial consortia are specified in Annex F. EDD developers are required to identify within each device details which profile has been used.

9.1.2 Syntax representation

The specification of the EDDL language in this section of the standard uses an abstract syntax. The abstract lexical structure in this section is converted into specific syntaxes specified in Annex C by the element name (e.g. 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.

9.1.3 EDD language elements

The language is structured in the following language elements:

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

Frequently used text strings e.g. help text and multi-lingual lists of labels should be separated in a “Text Dictionary” (see 9.27).

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.

9.1.4 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 sub-attributes, 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 due 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.

- **BLOCK_A** is a logical grouping of CHARACTERISTICS, PARAMETERS, PARAMETER_LISTS, and ITEM_LISTS (see Figure 27). To access one item of BLOCK_A, the instance of the block should be used (see 9.3.1 and Figure 27).

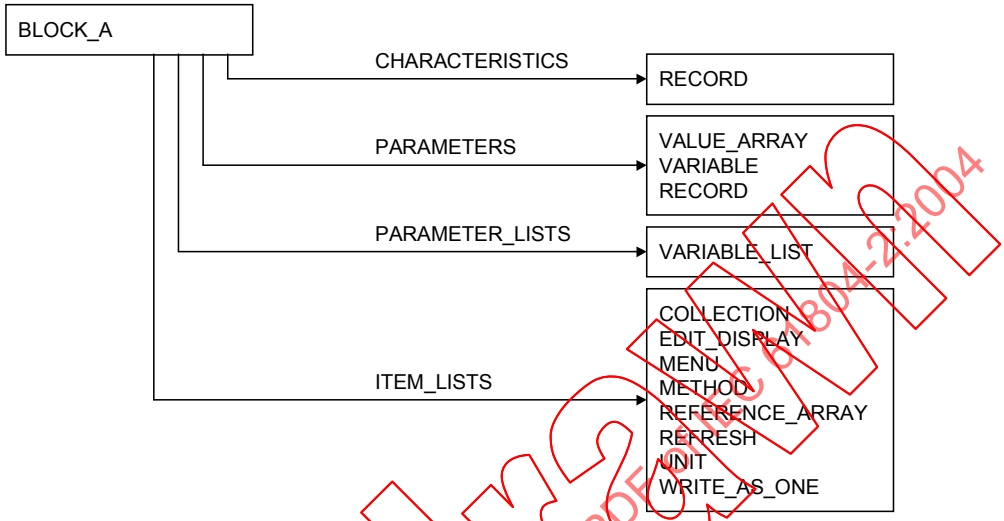


Figure 27 – BLOCK_A

- **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 9.3.2).

- **COLLECTION** describes logical groupings of data (see 9.4 and Figure 28).

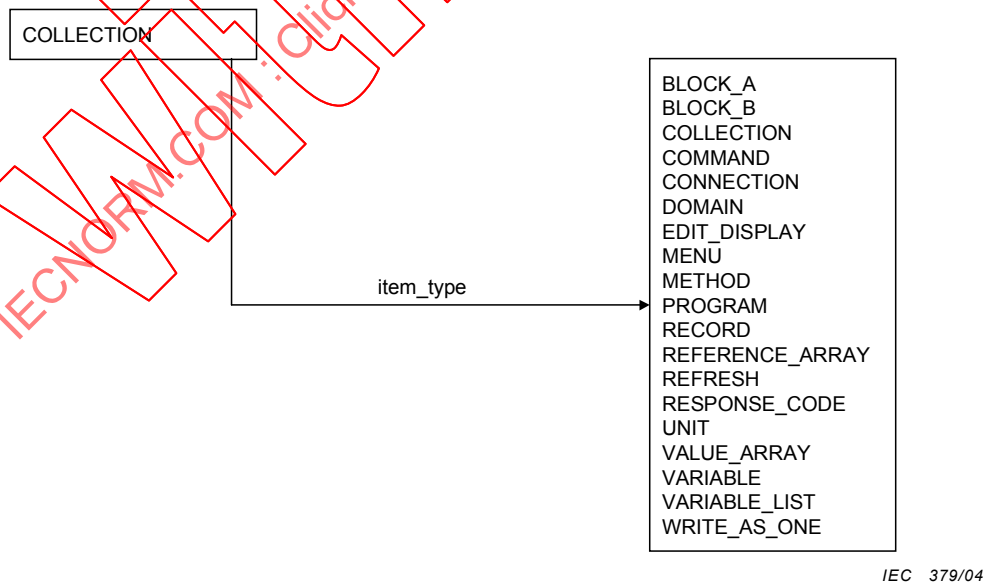


Figure 28 – COLLECTION

- **COMMAND** describes the structure and the addressing of the variables in the device (see 9.5 and Figure 29).

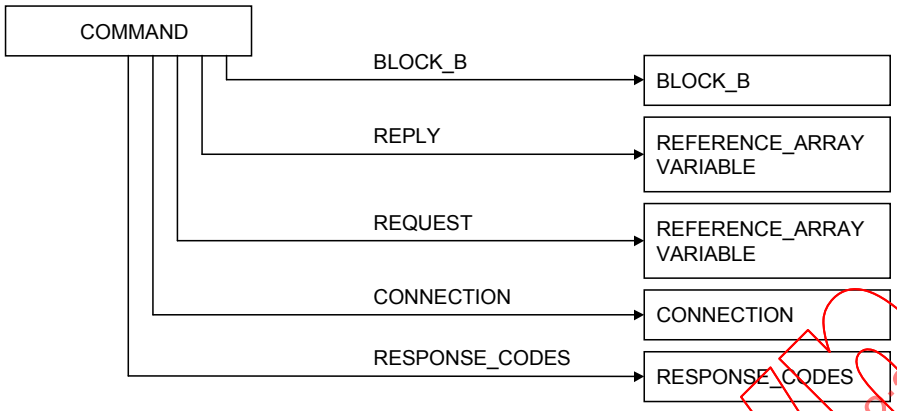


Figure 29 – COMMAND

- **CONNECTION** defines multiple applications in a device (see 9.6).
- **DOMAIN** can be used to download or upload moderately large amounts of data to or from a device (see 9.7 and Figure 30).

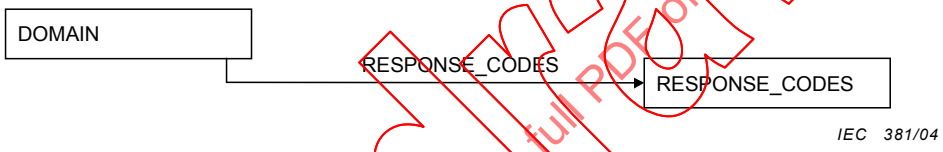


Figure 30 – DOMAIN

- **EDIT_DISPLAY** describes how the data will be represented to a user by a display device (see 9.8 and Figure 31).

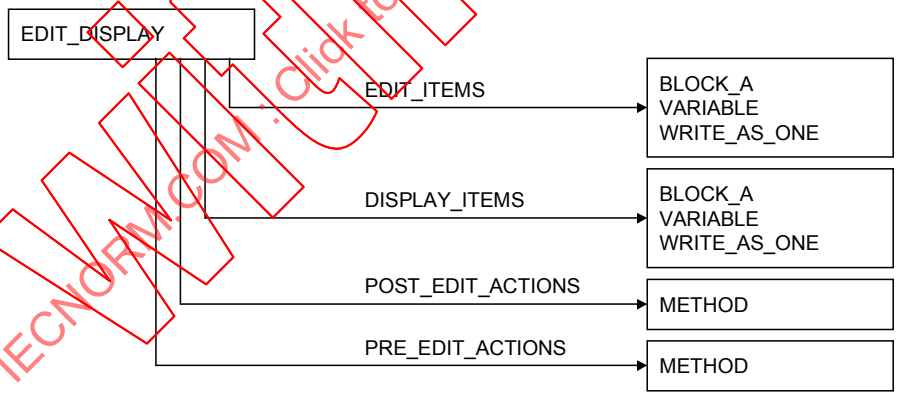


Figure 31 – EDIT_DISPLAY

- **IMPORT** is used to import an EDD and to modify its existing definitions (see 9.9).
- **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 9.10 and Figure 32).

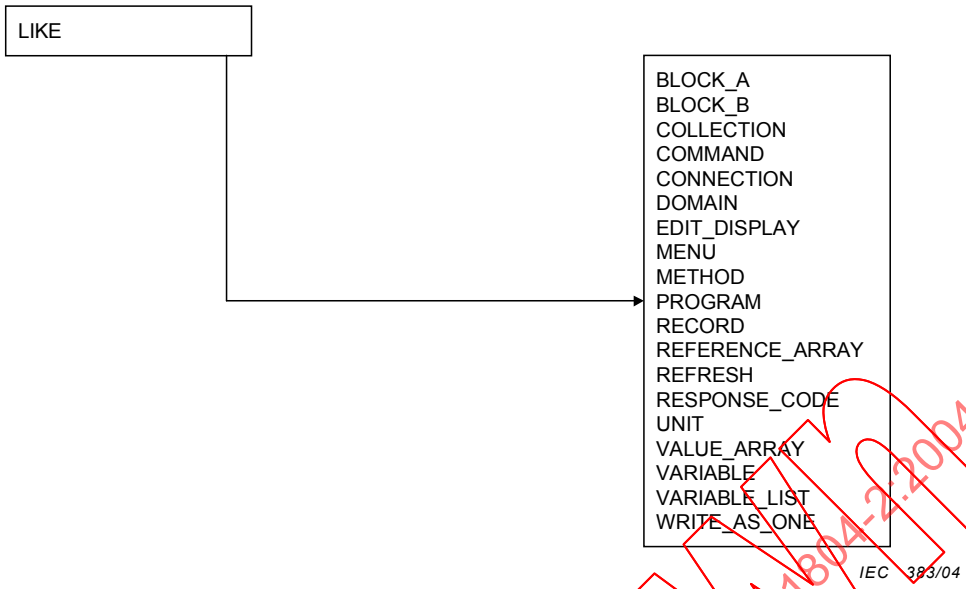


Figure 32 – LIKE

- **MENU** describes how the data will be presented to a user by an EDD application (see 9.11 and Figure 33).

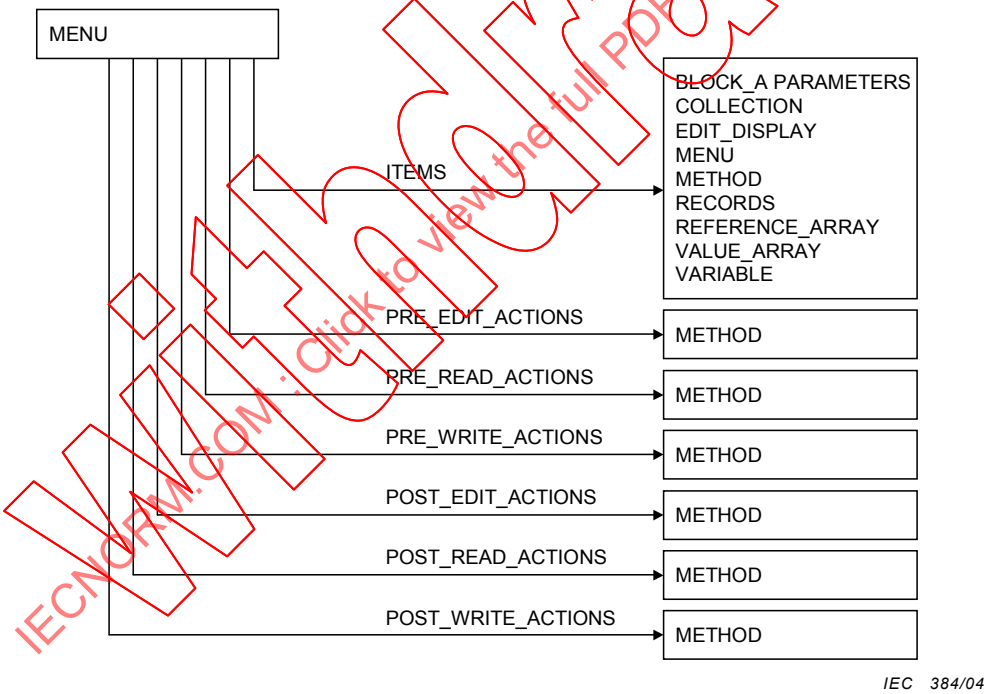


Figure 33 – MENU

- **METHOD** describes the execution of complex sequence of event interactions that must take place between system devices such as display units, configurators and field devices (see 9.12).
- **PROGRAM** specifies how device executable code can be initiated by an appropriate agent (see 9.13 and Figure 34).

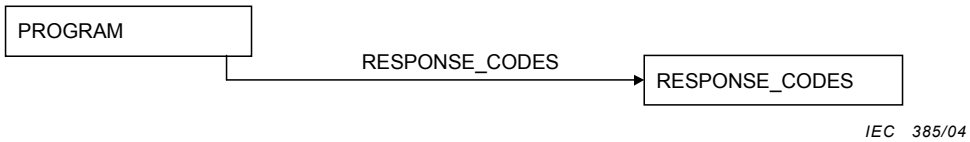


Figure 34 – PROGRAM

- **RECORD** describes the data contained in the device (see 9.14 and Figure 35).

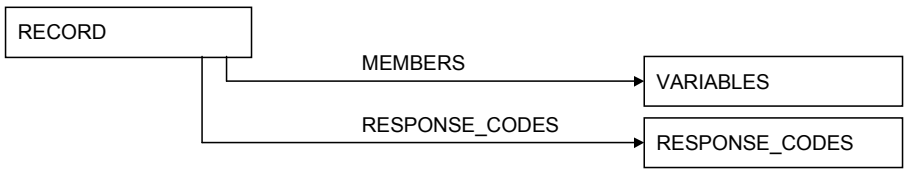


Figure 35 – RECORD

- **REFERENCE_ARRAY** is a set of EDD items of the same EDDL item type (e.g. VARIABLE or MENU). An item can be referenced in the EDD via the REFERENCE_ARRAY name and the index associated with the item (see 9.15 and Figure 36).

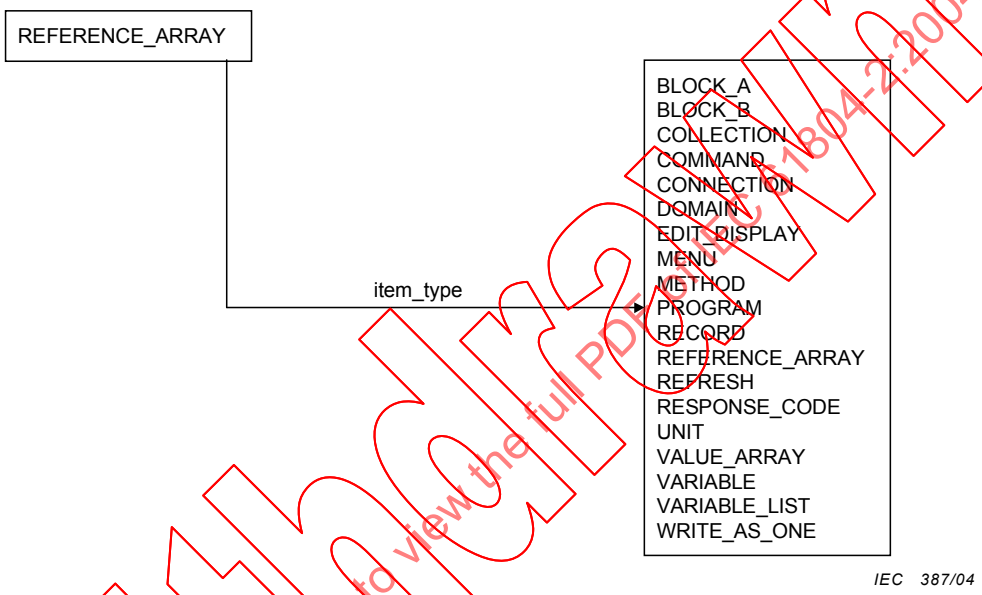


Figure 36 – REFERENCE_ARRAY

- **Relation:** three types of relations are specified.
  - REFRESH describes relationships between VARIABLEs, RECORDs, and VALUE_ARRAYs (see 9.16.2 and Figure 37).

Figure 37 – REFRESH

  - UNIT describes relationships between VARIABLEs, RECORDs, and VALUE_ARRAYs (see 9.16.3 and Figure 38).

Figure 38 – UNIT

— WRITE_AS_ONE RELATION describes relationships between VARIABLEs, RECORDs, and VALUE_ARRAYs (see 9.16.4 and Figure 39).

Figure 39 – 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 9.17).
- **VALUE_ARRAY** describes logical groups of EDDL variables (see 9.18 and Figure 40).

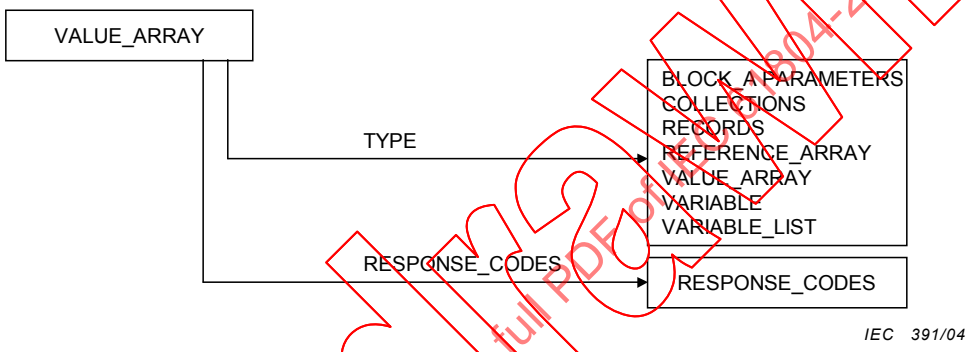


Figure 40 – VALUE_ARRAY

- **VARIABLE** describes the data contained in the device (see 9.19 and Figure 41).

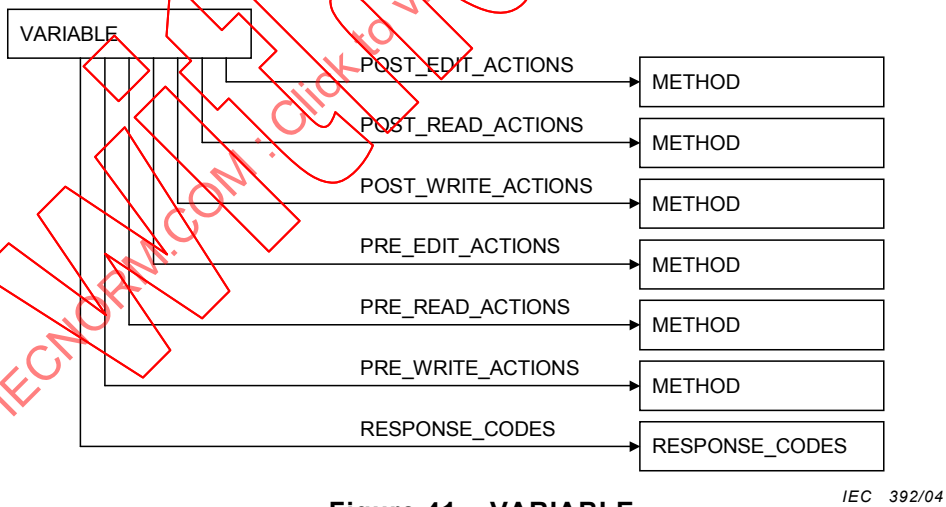


Figure 41 – VARIABLE

- **VARIABLE_LIST** describes logical groupings of data contained in the device that may be communicated as a list (see 9.20 and Figure 42).

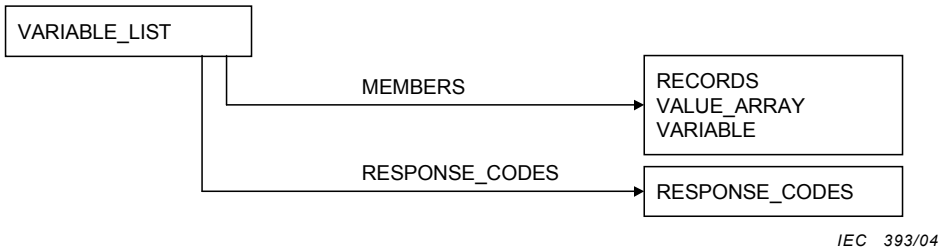


Figure 42 – VARIABLE_LIST

9.1.5 Common attributes

The following common attributes are used in many basic constructs:

- DEFINITION
- MEMBERS
- LABEL
- HELP
- RESPONSE_CODES

9.1.6 Special elements

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

- **Output Redirection**
  - — **OPEN** is used to create a separate output file for an EDD.
  - — **CLOSE** is used to close an opened output file for an EDD.
- **Conditional Expressions** specify attributes values, which are dependent on the values of variables at runtime
- **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 runtime

9.1.7 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.

9.1.8 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.

9.2 EDD identification information

9.2.1 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.

9.2.2 Specific attributes

9.2.2.1 DD_REVISION

Purpose

This 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 7.

Table 7 – DD_REVISION attribute

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

9.2.2.2 DEVICE_REVISION

DEVICE_REVISION
Purpose

This 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 8.

Table 8 – DEVICE_REVISION attribute

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

9.2.2.3 DEVICE_TYPE

Purpose

This 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 9.

Table 9 – DEVICE_TYPE attribute

Usage	Attribute	Description
s	integer	is a two-octet unsigned integer containing the code for a specific device type or group of device types
s	identifier	is 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.		

9.2.2.4 EDD_PROFILE

Purpose

This attribute specifies the profile of the EDDL as defined in Annex F of this document.

Lexical structure

```
EDD_PROFILE integer
```

The attribute of EDD_PROFILE is specified in Table 10.

Table 10 – EDD_PROFILE attribute

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

9.2.2.5 EDD_VERSION

Purpose

This keyword 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 11.

Table 11 – EDD_VERSION attribute

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

9.2.2.6 MANUFACTURER

Purpose

This 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)<exp>]
```

The attribute of MANUFACTURER is specified in Table 12.

Table 12 – MANUFACTURER attribute

Usage	Attribute	Description
s	long integer	is 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
sm	identifier	is 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.		

9.2.2.7 MANUFACTURER_EXT

Purpose

This attribute is reserved for future use. It is a placeholder for a string providing global identification for 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 13.

Table 13 – MANUFACTURER_EXT attribute

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

9.3 BLOCK

9.3.1 BLOCK_A

9.3.1.1 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 [CHARACTERICS, LABEL, PARAMETERS, COLLECTION_ITEMS,
EDIT_DISPLAY_ITEMS, HELP, MENU_ITEMS, METHOD_ITEMS, PARAMETER_LISTS,
REFERENCE_ARRAY_ITEMS, REFRESH_ITEMS, UNIT_ITEMS, WRITE_AS_ONE_ITEMS]
```

The attributes of BLOCK_A are specified in Table 14.

Table 14 – BLOCK_A attributes

Usage	Attribute	Description
m	CHARACTERISTICS	lexical element (see 9.3.1.2.1)
m	LABEL	lexical element (see 9.21.3)
m	PARAMETERS	lexical element (see 9.3.1.2.2)
o	COLLECTION_ITEMS	lexical element (see 9.3.1.2.3)
o	EDIT_DISPLAY_ITEMS	lexical element (see 9.3.1.2.4)
o	HELP	lexical element (see 9.21.2)
o	MENU_ITEMS	lexical element (see 9.3.1.2.5)
o	METHOD_ITEMS	lexical element (see 9.3.1.2.6)
o	PARAMETER_LISTS	lexical element (see 9.3.1.2.7)
o	REFERENCE_ARRAY_ITEMS	lexical element (see 9.3.1.2.8)
o	REFRESH_ITEMS	lexical element (see 9.3.1.2.9)
o	UNIT_ITEMS	lexical element (see 9.3.1.2.10)
o	WRITE_AS_ONE_ITEMS	lexical element (see 9.3.1.2.11)

9.3.1.2 Specific attributes

9.3.1.2.1 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 15.

Table 15 – CHARACTERISTIC attribute

Usage	Attribute	Description
m	reference	is a reference to a RECORD instance

9.3.1.2.2 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 16.

Table 16 – PARAMETER attributes

Usage	Attribute	Description
m	identifier	is 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	is a reference to a VARIABLE, VALUE_ARRAY or RECORD instance
o	description	is a short description for the reference
o	help	specifies help text for the reference

9.3.1.2.3 COLLECTION_ITEMS

Purpose

The COLLECTION_ITEMS list specifies the COLLECTIONs 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	is a reference to a COLLECTION instance

9.3.1.2.4 EDIT_DISPLAY_ITEMS

Purpose

The EDIT_DISPLAY_ITEMS list specifies the EDIT_DISPLAYs 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	is a reference to an EDIT_DISPLAY instance

9.3.1.2.5 MENU_ITEMS

Purpose

The MENU_ITEMS list specifies the MENUs associated with the BLOCK_A.

Lexical structure
`MENU_ITEMS reference+`

The attribute of MENU_ITEMS is specified in Table 19.

Table 19 – MENU_ITEMS attribute

Usage	Attribute	Description
m	reference	is a reference to a MENU instance

9.3.1.2.6 METHOD_ITEMS

Purpose

The METHOD_ITEMS list specifies the METHODs associated with the BLOCK_A.

Lexical structure

```
METHOD_ITEMS reference+
```

The attribute of METHOD_ITEMS is specified in Table 20.

Table 20 – METHOD_ITEMS attribute

Usage	Attribute	Description
m	reference	is a reference to a METHOD instance

9.3.1.2.7 PARAMETER_LISTS

Purpose

The PARAMETER_LISTS 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 21.

Table 21 – PARAMETER_LISTS attributes

Usage	Attribute	Description
m	identifier	is 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	is a reference to a VARIABLE_LIST instance
o	description	is a short description of the item
o	help	specifies help text for the item

9.3.1.2.8 REFERENCE_ARRAY_ITEMS

Purpose

The REFERENCE_ARRAY_ITEMS list 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 22.

Table 22 – REFERENCE_ARRAY_ITEMS attribute

Usage	Attribute	Description
m	reference	is a reference to a REFERENCE_ARRAY instance

9.3.1.2.9 REFRESH_ITEMS

Purpose

The REFRESH_ITEMS list specifies the REFRESH relations associated with BLOCK_A.

Lexical structure
```
REFRESH_ITEMS reference+
```
The attribute of REFRESH_ITEMS is specified in Table 23.

Table 23 – REFRESH_ITEMS attribute

Usage	Attribute	Description
m	reference	is a reference to a REFRESH instance

9.3.1.2.10 UNIT_ITEMS

Purpose

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

Lexical structure
```
UNIT_ITEMS reference+
```
The attribute of UNIT_ITEMS is specified in Table 24.

Table 24 – UNIT_ITEMS attribute

Usage	Attribute	Description
m	reference	is a reference to an UNIT instance

9.3.1.2.11 WRITE_AS_ONE_ITEMS

Purpose

The WRITE_AS_ONE_ITEMS list 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 25.

Table 25 – WRITE_AS_ONE_ITEMS attribute

Usage	Attribute	Description
m	reference	is a reference to a WRITE_AS_ONE instance

9.3.2 BLOCK_B

9.3.2.1 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 26.

Table 26 – BLOCK_B attributes

Usage	Attribute	Description
m	NUMBER	lexical element (see Table 27)
m	TYPE	lexical element (see Table 28)

9.3.2.2 Specific attributes

9.3.2.2.1 NUMBER

Purpose

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

Lexical structure

```
NUMBER (integer, expression)<exp>
```

The attribute of NUMBER is specified in Table 27.

Table 27 – NUMBER attributes

Usage	Attribute	Description
s	integer	is the number of the BLOCK_B instance of the same TYPE
s	expression	is an expression, which shall be evaluate to a non-negative integer that is within the index range of the NUMBER attribute (for the description of expressions, see 9.25.7).

9.3.2.2.2 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 28.

Table 28 – TYPE attributes

Usage	Attribute	Description
s	FUNCTION	a named block consisting of one or more input, output and contained parameters. FBs represent the basic automation functions performed by an EDD application, which is independent of the specific devices and the network. Each FB 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 FB EDD application or by other FB applications
s	PHYSICAL	hardware/resource specific characteristics of a field device are made visible through the physical block. Similar to transducer blocks, they isolate FBs from the physical hardware by containing a set of implementation independent hardware/resource parameters

Usage	Attribute	Description
s	TRANSDUCER	transducer blocks isolate FBs 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 FBs. Transducer blocks perform functions, such as calibration and linearization, on I/O data to convert it to a device-independent representation

9.4 COLLECTION

9.4.1 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, e.g. to identify parameters that should be processed together.

NOTE 2 The recommended supported nesting level is two (e.g. COLLECTION of COLLECTION).

Lexical structure

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

The attributes of COLLECTION are specified in Table 29.

Table 29 – COLLECTION attributes

Usage	Attribute	Description
m	item-type	lexical element (see Table 30)
m	MEMBERS	lexical element (see 9.21.4)
o	HELP	lexical element (see 9.21.2)
o	LABEL	lexical element (see 9.21.3)

9.4.2 Specific attributes - item-type

Purpose

The item-type specifies the type of members in the collection. All the collection members shall be of the specified type.

Lexical structure

```
item-type
```

Table 30 shows the allowed item-types:

Table 30 – item-type

Usage	item-type	Description
s	BLOCK_A	selection of this basic construct
s	BLOCK_B	selection of this basic construct
s	COLLECTION	selection of this basic construct
s	COMMAND	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	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

Usage	item-type	Description
s	REFERENCE_ARRAY	selection of this basic construct
s	REFRESH	selection of this basic construct
s	RESPONSE_CODES	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	WRITE_AS_ONE	selection of this basic construct

9.5 COMMAND

9.5.1 General structure

Purpose

The 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 9.11.2.10).

NOTE 1 The address field of a communication frame is specified using the selectable constructs BLOCK / SLOT / NUMBER together with INDEX and 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 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 / 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,
CONNECTION, HEADER, MODULE, RESPONSE_CODES]
```

The attributes of COMMAND are specified in Table 31.

Table 31 – COMMAND attributes

Usage	Attribute	Description
m	OPERATION	lexical element (see 9.5.2.1)
m	TRANSACTION	lexical element (see 9.5.2.2)
c	INDEX	lexical element (see 9.5.2.3)
s	BLOCK_B	lexical element (see 9.5.2.4)
s	NUMBER	lexical element (see 9.5.2.5)
s	SLOT	lexical element (see 9.5.2.6)
o	CONNECTION	lexical element (see 9.5.2.7)
o	HEADER	lexical element (see 9.5.2.8)
o	MODULE	lexical element (see 9.5.2.9)
o	RESPONSE_CODES	lexical element (see 9.21.5)

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.

9.5.2 Specific attributes

9.5.2.1 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)<exp>]
```

The attributes of OPERATION are specified in Table 32.

Table 32 – OPERATION attribute

Usage	Attribute	Description
s	COMMAND	specifies that the device performs a predefined set of actions (e.g. 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

9.5.2.2 TRANSACTION

9.5.2.2.1 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 HART[5] Universal Command Revision 4 or later, are examples of multiple transaction commands.

Lexical structure

```
TRANSACTION [REPLY, REQUEST, integer, RESPONSE_CODES]
```

The attributes of TRANSACTION are specified in Table 33.

Table 33 – TRANSACTION attributes

Usage	Attribute	Description
m	REPLY	lexical element (see 9.5.2.2.2.1)
m	REQUEST	lexical element (see 9.5.2.2.2.2)
o	integer	specifies the number of the TRANSACTIONS (if more than one TRANSACTION is used)
o	RESPONSE_CODES	lexical element (see 9.17)

[5] See Clause F.4.

9.5.2.2.2 Specific attributes

9.5.2.2.2.1 REPLY

Purpose

The REPLY attribute contains a list of data items (constants or references to variables), which are received from the device. The order of the list shall be maintained in the corresponding data field in the communicated PDU.

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.

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 E.

Lexical structure

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

The attributes of REPLY are specified in Table 34.

Table 34 – REPLY and REQUEST attributes

Usage	Attribute	Description
m	reference	is a reference to a VARIABLE instance
o	item-mask	specifies a bit pattern to extract the VARIABLE from the data field 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	specifies that the VARIABLE is not actually stored in the device. It provides additional information to ensure correct processing
o	INDEX	specifies that 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.

9.5.2.2.2.2 REQUEST

Purpose

The REQUEST attribute contains a list of data items (constants or references to variables), which are sent to the device. The order of the list shall be maintained in the corresponding data field in the communicated PDU.

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 E.

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

The attributes of REQUEST are specified in Table 34.

9.5.2.3 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 35.

Table 35 – INDEX attribute

Usage	Attribute	Description
s	integer	is the number of the index
s	reference	is a reference to a VARIABLE instance containing the index
s	expression	is 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 9.25.7).

9.5.2.4 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 36.

Table 36 – BLOCK_B attribute

Usage	Attribute	Description
m	reference	is a reference to a BLOCK_B instance

9.5.2.5 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 37.

Table 37 – NUMBER attribute

Usage	Attribute	Description
s	integer	is the number of the COMMAND
s	reference	is a reference to a VARIABLE instance containing the number of the COMMAND
s	expression	is 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 9.25.7).

9.5.2.6 SLOT

Purpose

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

Lexical structure

```
SLOT (integer, reference, expression)<exp>
```

The attributes of SLOT are specified in Table 38.

Table 38 – SLOT attribute

Usage	Attribute	Description
s	integer	is the number of the slot
s	reference	is a reference to a VARIABLE instance containing the slot number
s	expression	is 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 9.25.7).

9.5.2.7 CONNECTION

Purpose

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

Lexical structure

```
CONNECTION (reference)<exp>
```

The attribute of CONNECTION is specified in Table 39.

Table 39 – CONNECTION attribute

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

9.5.2.8 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 40.

Table 40 – HEADER attribute

Usage	Attribute	Description
m	string	specifies a string that is used for the communication with specific communication protocols

9.5.2.9 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 41.

Table 41 – MODULE attribute

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

9.6 CONNECTION

9.6.1 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 42.

Table 42 – CONNECTION attribute

Usage	Attribute	Description
m	APPINSTANCE	lexical element (see 9.6.2)

9.6.2 Specific attribute - APPINSTANCE

Purpose

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 43.

Table 43 – APPINSTANCE attribute

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

9.7 DOMAIN

9.7.1 General structure

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_CODE]
```

The attributes of DOMAIN are specified in Table 44.

Table 44 – DOMAIN attributes

Usage	Attribute	Description
o	HANDLING	lexical element (see 9.7.2)
o	RESPONSE_CODES	lexical element (see 9.21.5)

9.7.2 Specific attribute - HANDLING

Purpose

The HANDLING attribute specifies the operations, which can be performed on the DOMAIN. By default, a DOMAIN 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 45.

Table 45 – HANDLING attribute

Usage	Attribute	Description
s	READ	indicates that the DOMAIN memory contents can only be read from the device
s	READ_WRITE	indicates that the DOMAIN memory contents can be read from and written to the device
s	WRITE	indicates that the DOMAIN memory contents can only be written to the device

9.8 EDIT_DISPLAY

9.8.1 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 46.

Table 46 – EDIT_DISPLAY attributes

Usage	Attribute	Description
M	EDIT_ITEMS	lexical element (see 9.8.2.1)
M	LABEL	lexical element (see 9.21.3)
O	DISPLAY_ITEMS	lexical element (see 9.8.2.2)
O	POST_EDIT_ACTIONS	lexical element (see 9.8.2.3)
O	PRE_EDIT_ACTIONS	lexical element (see 9.8.2.4)

9.8.2 Specific attributes

9.8.2.1 EDIT_ITEMS

Purpose

EDIT_ITEMS 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 (e.g. 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 47.

Table 47 – EDIT_ITEMS attribute

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

9.8.2.2 DISPLAY_ITEMS

Purpose

DISPLAY_ITEMS defines the set of data items 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 (e.g. 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 48.

Table 48 – DISPLAY_ITEM attributes

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

9.8.2.3 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.

Lexical structure

```
POST_EDIT_ACTIONS [(reference)<exp>]+, DEFINITION
```

The attributes of POST_EDIT_ACTIONS are specified in Table 49.

9.8.2.4 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.

Lexical structure

```
PRE_EDIT_ACTIONS [(reference)<exp>]+, DEFINITION
```

The attributes of PRE_EDIT_ACTIONS are specified in Table 49.

Table 49 – POST_EDIT_ACTIONS, PRE_EDIT_ACTIONS attribute

Usage	Attribute	Description
m	reference	is a reference to a METHOD instance
o	DEFINITION	lexical element (see 9.21.1)

9.9 IMPORT

9.9.1 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 43 shows the EDDL import mechanisms.

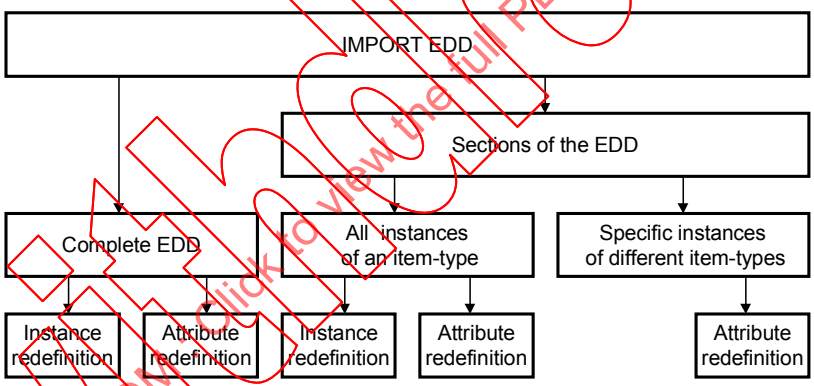


Figure 43 – EDDL import mechanisms

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,
(BLOCKS_A, BLOCKS_B, COLLECTIONS, COMMANDS, CONNECTIONS, DOMAINS,
EDIT_DISPLAYS, MENUS, METHODS, PROGRAMS, RECORDS, REFERENCE_ARRAYS,
REFRESHS, RELATIONS, RESPONSE_CODES, UNITS, VALUE_ARRAYS, VARIABLES,
VARIABLE_LISTS, WRITE_AS_ONES,
reference*,
[DELETE identifier]*,
[REDEFINE identifier]*,
attribute-redefinition*))
```

The attributes are specified in Table 50.

Table 50 – Importing Device Description

Usage	Attribute	Description
o	attribute-redefinition	(see 9.9.2)
m	DD_REVISION	(see 9.2.2.1)
m	DEVICE_REVISION	(see 9.2.2.2)
m	DEVICE_TYPE	(see 9.2.2.3)
o	EDD_PROFILE	(see 9.2.2.4)
o	EDD_VERSION	(see 9.2.2.5)
o	FILE_NAME	specifies 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
m	MANUFACTURER	(see 9.2.2.6)
c	MANUFACTURER_EXT	(see 9.2.2.7)
o	EVERYTHING	specifies that 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	BLOCKS_A	specifies that all BLOCK_A instances of an external EDD are imported
o	BLOCKS_B	specifies that all BLOCK_B instances of an external EDD are imported
o	COLLECTIONS	specifies that all COLLECTION instances of an external EDD are imported
o	COMMANDS	specifies that all COMMAND instances of an external EDD are imported
o	CONNECTIONS	specifies that all CONNECTION instances of an external EDD are imported
o	DOMAINS	specifies that all DOMAIN instances of an external EDD are imported
o	EDIT_DISPLAYS	specifies that all EDIT_DISPLAY instances of an external EDD are imported
o	MENUS	specifies that all MENU instances of an external EDD are imported
o	METHODS	specifies that all METHOD instances of an external EDD are imported
o	PROGRAMS	specifies that all PROGRAM instances of an external EDD are imported
o	RECORDS	specifies that all RECORD instances of an external EDD are imported
o	REFERENCE_ARRAYS	specifies that all REFERENCE_ARRAY instances of an external EDD are imported
o	REFRESHES	specifies that all REFRESH instances of an external EDD are imported
o	RELATIONS	specifies that all REFRESH, UNIT and WRITE_AS_ONE instances of an external EDD are imported
o	RESPONSE_CODES	specifies that all RESPONSE_CODE instances of an external EDD are imported
o	UNITS	specifies that all UNIT instances of an external EDD are imported
o	VALUE_ARRAYS	specifies that all VALUE_ARRAY instances of an external EDD are imported
o	VARIABLES	specifies that all VARIABLE instances of an external EDD are imported
o	VARIABLE_LISTS	specifies that all VARIABLE_LIST instances of an external EDD are imported
o	WRITE_AS_ONES	specifies that all WRITE_AS_ONE instances of an external EDD are imported
o	reference	specifies that a single instance is imported. A single instance shall not be imported, if it is already included in an item type import
o	DELETE	specifies that an imported instance shall be deleted
o	REDEFINE	specifies that all attributes of an imported instance shall be redefined

9.9.2 Specific attributes – attribute-redefinition

Purpose

The 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.

Lexical structure – BLOCK_A

```
BLOCK identifier, [[ADD, DELETE, REDEFINE], [CHARACTERISTICS, LABEL,
PARAMETERS, parameters-list-element, COLLECTION_ITEMS, EDIT_DISP_ITEMS,
HELP, MENU_ITEMS, METHOD_ITEMS, PARAMETER_LISTS, parameters-lists-list-
element, REFERENCE_ARRAY_ITEMS, REFRESH_ITEMS, UNIT_ITEMS,
WRITE_AS_ONE_ITEMS]]+
```

The attributes are specified in Table 51 and in Table 52.

Table 51 – 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 52 to Table 67)
s	DELETE	deletes the following specified attribute. "D" indicates that DELETE can be applied to the attribute in the redefinition rules tables (see Table 52 to Table 67)
s	REDEFINE	redefine the following specified attribute. "R" indicates that REDEFINE can be applied to the attribute in the redefinition rules tables (see Table 52 to Table 67)

The redefinition rules of BLOCK_A are specified in Table 52.

Table 52 – Redefinition rules for BLOCK_A attributes

A	D	R	Attribute	Description
		•	CHARACTERISTICS	lexical element (see 9.3.1.2.1)
		•	LABEL	lexical element (see 9.21.3)
		•	PARAMETERS	lexical element (see 9.3.1.2.2)
•	•	•	parameters-list-element	single enumeration list elements of PARAMETERS may be changed
	•	•	COLLECTION_ITEMS	lexical element (see 9.3.1.2.3)
	•	•	EDIT_DISP_ITEMS	lexical element (see 9.3.1.2.4)
	•	•	HELP	lexical element (see 9.21.2)
	•	•	MENU_ITEMS	lexical element (see 9.3.1.2.5)
	•	•	METHOD_ITEMS	lexical element (see 9.3.1.2.6)
		•	PARAMETER_LISTS	lexical element (see 9.3.1.2.7)
•	•	•	parameters-lists-list-element	single enumeration list elements of PARAMETERS_LISTS may be changed
	•	•	REFERENCE_ARRAY_ITEMS	lexical element (see 9.3.1.2.8)
	•	•	REFRESH_ITEMS	lexical element (see 9.3.1.2.9)
	•	•	UNIT_ITEMS	lexical element (see 9.3.1.2.10)
	•	•	WRITE_AS_ONE_ITEMS	lexical element (see 9.3.1.2.11)

Lexical structure – BLOCK_B

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

The redefinition rules of BLOCK_B are specified in Table 53 and Table 51.

Table 53 – Redefinition rules for BLOCK_B attributes

A	D	R	Attribute	Description
		•	NUMBER	lexical element (see 9.3.2.2.1)
		•	TYPE	lexical element (see 9.3.2.2.2)

Lexical structure — COLLECTION
`COLLECTION identifier, [[ADD, DELETE, REDEFINE], [MEMBERS, members-list-element, HELP, LABEL]]+`

The redefinition rules of COLLECTION are specified in Table 54 and Table 51.

Table 54 – Redefinition rules for COLLECTION attributes

A	D	R	Attribute	Description
		•	MEMBERS	lexical element (see 9.21.4)
•	•	•	members-list-element	single enumeration list elements of MEMBERS may be changed
	•	•	HELP	lexical element (see 9.21.2)
	•	•	LABEL	lexical element (see 9.21.3)

Lexical structure — COMMAND
`COMMAND identifier, [[ADD, DELETE, REDEFINE], [OPERATION, TRANSACTION, INDEX, BLOCK_B, NUMBER, SLOT, CONNECTION, HEADER, MODULE, RESONSE_CODES, response-codes-list-element]]+`

The redefinition rules of COMMAND are specified in Table 55 and Table 51.

Table 55 – Redefinition rules for COMMAND attributes

A	D	R	Attribute	Description
		•	OPERATION	lexical element (see 9.5.2.1)
		•	TRANSACTION	lexical element (see 9.5.2.2)
		•	INDEX	lexical element (see 9.5.2.3)
		•	BLOCK_B	lexical element (see 9.5.2.4)
		•	NUMBER	lexical element (see 9.5.2.5)
		•	SLOT	lexical element (see 9.5.2.6)
	•	•	CONNECTION	lexical element (see 9.5.2.7)
	•	•	HEADER	lexical element (see 9.5.2.8)
	•	•	MODULE	lexical element (see 9.5.2.9)
	•	•	RESPONSE_CODES	lexical element (see 9.21.5)
•	•	•	response-codes-list-element	single enumeration list elements of RESPONSE_CODE may be changed

Lexical structure — CONNECTION
`CONNECTION identifier, [REDEFINE, [APPINSTANCE]]+`

The redefinition rules of CONNECTION are specified in Table 56and Table 51.

Table 56 – Redefinition rules for CONNECTION attributes

A	D	R	Attribute	Description
		•	APPINSTANCE	lexical element (see 9.6.2)

Lexical structure — DOMAIN
`DOMAIN identifier, [DELETE, REDEFINE], [HANDLING, RESPONSE_CODES]+`

The redefinition rules of DOMAIN are specified in Table 57 and Table 51.

Table 57 – Redefinition rules for DOMAIN attributes

A	D	R	Attribute	Description
	•	•	HANDLING	lexical element (see 9.7.2)
	•	•	RESPONSE_CODES	lexical element (see 9.21.5)

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 58 and Table 51.

Table 58 – Redefinition rules for EDIT_DISPLAY attributes

A	D	R	Attribute	Description
		•	EDIT_ITEMS	lexical element (see 9.8.2.1)
		•	LABEL	lexical element (see 9.21.3)
	•	•	DISPLAY_ITEMS	lexical element (see 9.8.2.2)
	•	•	POST_EDIT_ACTIONS	lexical element (see 9.8.2.3)
	•	•	PRE_EDIT_ACTIONS	lexical element (see 9.8.2.4)

Lexical structure – MENU
`MENU identifier, [[ADD, DELETE, REDEFINE], [ITEMS, items-list-element, LABEL, ACCESS, ENTRY, 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 59 and Table 51.

Table 59 – Redefinition rules for MENU attributes

A	D	R	Attribute	Description
		•	ITEMS	lexical element (see 9.11.2.1)
•	•	•	items-list-element	single enumeration list elements of ITEMS be changed
		•	LABEL	lexical element (see 9.21.3)
	•	•	ACCESS	lexical element (see 9.12.2.1)
	•	•	ENTRY	lexical element (see 9.11.2.3)
	•	•	HELP	lexical element (see 9.21.2)
	•	•	POST_EDIT_ACTIONS	lexical element (see 9.11.2.4)
	•	•	POST_READ_ACTIONS	lexical element (see 9.11.2.5)
	•	•	POST_WRITE_ACTIONS	lexical element (see 9.11.2.6)
	•	•	PRE_EDIT_ACTIONS	lexical element (see 9.11.2.7)
	•	•	PRE_READ_ACTIONS	lexical element (see 9.11.2.8)

A	D	R	Attribute	Description
	•	•	PRE_WRITE_ACTIONS	lexical element (see 9.11.2.9)
	•	•	PURPOSE	lexical element (see 9.11.2.10)
	•	•	ROLE	lexical element (see 9.11.2.11)
	•	•	STYLE	lexical element (see 9.11.2.12)
	•	•	VALIDITY	lexical element (see 9.11.2.13)

Lexical structure — METHOD

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

The redefinition rules of METHOD are specified in Table 60 and Table 51.

Table 60 – Redefinition rules for METHOD attributes

A	D	R	Attribute	Description
	•	•	ACCESS	lexical element (see 9.12.2.1)
		•	CLASS	lexical element (see 9.12.2.2)
		•	DEFINITION	lexical element (see 9.21.1)
		•	LABEL	lexical element (see 9.21.3)
	•	•	HELP	lexical element (see 9.21.2)
	•	•	VALIDITY	lexical element (see 9.12.2.3)

Lexical structure — PROGRAM

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

The redefinition rules of PROGRAM are specified in Table 61 and Table 51.

Table 61 – Redefinition rules for PROGRAM attributes

A	D	R	Attribute	Description
		•	ARGUMENT	lexical element (see 9.13.2)
	•	•	argument-list-element	single enumeration list elements of ARGUMENTS may be changed
	•	•	RESPONSE_CODES	lexical element (see 9.21.5)

Lexical structure — RECORD

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

The redefinition rules of RECORD are specified in Table 62 and Table 51.

Table 62 – Redefinition rules for RECORD attributes

A	D	R	Attribute	Description
		•	MEMBERS	lexical element (see 9.21.4)
•	•	•	members-list-element	single enumeration list elements of MEMBERS may be changed
	•	•	LABEL	lexical element (see 9.21.3)
	•	•	HELP	lexical element (see 9.21.2)
	•	•	RESPONSE_CODES	lexical element (see 9.21.5)

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

The redefinition rules of REFERENCE_ARRAY are specified in Table 63 and Table 51.

Table 63 – Redefinition rules for REFERENCE_ARRAY attributes

A	D	R	Attribute	Description
		•	ELEMENTS	lexical element (see 9.15.2)
•	•	•	elements-list-element	single enumeration list elements of ELEMENTS may be changed
	•	•	LABEL	lexical element (see 9.21.3)
	•	•	HELP	lexical element (see 9.21.2)

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

The redefinition rules of RESPONSE_CODES are specified in Table 64 and Table 51.

Table 64 – 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 – VALUE_ARRAY
VALUE_ARRAY identifier, [[DELETE, REDEFINE], [LABEL, NUMBER_OF_ELEMENTS, TYPE, HELP, RESPONSE_CODES]]+

The redefinition rules of VALUE_ARRAY are specified in Table 65 and Table 51.

Table 65 – Redefinition rules for VALUE_ARRAY attributes

A	D	R	Attribute	Description
		•	LABEL	lexical element (see 9.21.3)
		•	NUMBER_OF_ELEMENTS	lexical element (see 9.18.2.1)
		•	TYPE	lexical element (see 9.18.2.2)
	•	•	HELP	lexical element (see 9.21.2)
	•	•	RESPONSE_CODES	lexical element (see 9.21.5)

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, type-list-element, CONSTANT_UNIT, HANDLING, HELP, POST_EDIT_ACTIONS, POST_READ_ACTIONS, POST_WRITE_ACTIONS, PRE_EDIT_ACTION, PRE_READ_ACTIONS, PRE_WRITE_ACTIONS, READ_TIMEOUT, RESPONSE_CODES, STYLE, VALIDITY, WRITE_TIMEOUT]]+

The redefinition rules are specified in Table 66 and Table 51.

Table 66 – Redefinition rules for VARIABLE attributes

A	D	R	Attribute	Description
		•	CLASS	lexical element (see 9.19.2.1)
		•	LABEL	lexical element (see 9.21.3)
			TYPE	lexical element (see 9.19.2.2)
	•	•	DEFAULT_VALUE	lexical element (see 9.19.2.2)
	•	•	DISPLAY_FORMAT	lexical element (see 9.19.2.2)
	•	•	EDIT_FORMAT	lexical element (see 9.19.2.2)
	•	•	INITIAL_VALUE	lexical element (see 9.19.2.2)
	•	•	MAX_VALUE	lexical element (see 9.19.2.2)
	•	•	MIN_VALUE	lexical element (see 9.19.2.2)
	•	•	SCALING_FACTOR	lexical element (see 9.19.2.2)
•	•	•	type-list-element	single enumeration list elements of every TYPE may be changed
	•	•	CONSTANT_UNIT	lexical element (see 9.19.2.3)
	•	•	HANDLING	lexical element (see 9.19.2.4)
	•	•	HELP	lexical element (see 9.21.2)
	•	•	POST_EDIT_ACTIONS	lexical element (see 9.19.2.5)
	•	•	POST_READ_ACTIONS	lexical element (see 9.19.2.6)
	•	•	POST_WRITE_ACTIONS	lexical element (see 9.19.2.7)
	•	•	PRE_EDIT_ACTION	lexical element (see 9.19.2.8)
	•	•	PRE_READ_ACTIONS	lexical element (see 9.19.2.9)
	•	•	PRE_WRITE_ACTIONS	lexical element (see 9.19.2.10)
	•	•	READ_TIMEOUT	lexical element (see 9.19.2.11)
	•	•	RESPONSE_CODES	lexical element (see 9.21.5)
	•	•	STYLE	lexical element (see 9.19.2.12)
	•	•	VALIDITY	lexical element (see 9.19.2.13)
	•	•	WRITE_TIMEOUT	lexical element (see 9.19.2.11)

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 67 and Table 51.

Table 67 – Redefinition rules for VARIABLE_LIST attributes

A	D	R	Attribute	Description
		•	MEMBERS	lexical element (see 9.21.4)
•	•	•	members-list-element	single enumeration list elements of MEMBERS may be changed.
	•	•	HELP	lexical element (see 9.21.2)
	•	•	LABEL	lexical element (see 9.21.3)
	•	•	RESPONSE_CODES	lexical element (see 9.21.5)

9.10 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 68.

Table 68 – LIKE attributes

Usage	Attribute	Description
m	identifier	is the name of the new EDD instance
m	reference	is a reference to an EDD instance which shall be copied
o	attribute-redefinition	specifies whether an attribute is added, deleted or redefined (see 9.9.2). Only existing attributes can be deleted or redefined
o	item-type	can be one of the elements listed in Table 30. Even if item-type is present it is not interpreted

9.11 MENU

9.11.1 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. The last elements of a hierarchical MENU structure shall be methods, variables or basic constructs referencing groups of them (e.g. COLLECTION OF VARIABLES).

If a sub menu is displayed, the menu in which this sub menu is referenced shall be valid.

Lexical structure

```
MENU identifier [ITEM, LABEL, ACCESS, POST_EDIT_ACTIONS, POST_READ_ACTIONS,
POST_WRITE_ACTIONS, PRE_EDIT_ACTIONS, PRE_READ_ACTIONS, PRE_WRITE_ACTIONS,
STYLE, VALIDITY]
```

The attributes of MENU are specified in Table 69.

Table 69 – MENU attribute

Usage	Attribute	Description
m	ITEMS	lexical element (see 9.11.2.1)
m	LABEL	lexical element (see 9.21.3)
o	ACCESS	lexical element (see 9.11.2.2)
o	ENTRY	lexical element (see 9.11.2.3)
o	HELP	lexical element (see 9.21.2)
o	POST_EDIT_ACTIONS	lexical element (see 9.11.2.4)
o	POST_READ_ACTIONS	lexical element (see 9.11.2.5)
o	POST_WRITE_ACTIONS	lexical element (see 9.11.2.6)
o	PRE_EDIT_ACTIONS	lexical element (see 9.11.2.7)
o	PRE_READ_ACTIONS	lexical element (see 9.11.2.8)

Usage	Attribute	Description
o	PRE_WRITE_ACTIONS	lexical element (see 9.11.2.9)
c	PURPOSE	lexical element (see 9.11.2.10). The PURPOSE shall be specified if the ENTRY attribute is used
o	ROLE	lexical element (see 9.11.2.11)
o	STYLE	lexical element (see 9.11.2.12)
o	VALIDITY	lexical element (see 9.11.2.13)

9.11.2 Specific attributes

9.11.2.1 ITEMS

Purpose

The ITEMS attribute specifies selected elements (VARIABLE, BLOCK_A PARAMETERS, elements of BLOCK_A PARAMETERS, COLLECTION, EDIT_DISPLAY, METHOD, RECORDS, REFERENCE_ARRAY, VALUE_ARRAY, VARIABLE_LIST and other MENUs) of the EDD, which shall be displayed to the user plus optional qualifiers for each element. These qualifiers (DISPLAY_VALUE, HIDDEN, READ_ONLY, 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.

Lexical structure

```
ITEMS [(reference, DISPLAY_VALUE, HIDDEN, READ_ONLY, REVIEW)<exp>]+
```

The attributes of ITEMS are specified in Table 70.

Table 70 – ITEMS attribute

Usage	Attribute	Description
m	reference	is a reference to a VARIABLE, a BLOCK_A PARAMETERS, an element of BLOCK_A PARAMETERS, a COLLECTION, an EDIT_DISPLAY, a METHOD, RECORDS, a REFERENCE_ARRAY, a VALUE_ARRAY, a VARIABLE_LIST or a MENU instance
c/o	DISPLAY_VALUE	c 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	specifies that the item is not displayed
c/o	READ_ONLY	c only used with VARIABLE. Specifies that the value shall be displayed but it may not be modified (regardless of its HANDLING attribute)
c/o	REVIEW	c only used with MENU. REVIEW is used to present a large quantity of data, e.g. for review purpose

9.11.2.2 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 71.

Table 71 – ACCESS attribute

Usage	Attribute	Description
s	OFFLINE	specifies that the values of the items displayed in the MENU are read locally
s	ONLINE	specifies that 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).

9.11.2.3 ENTRY

Purpose

The ENTRY attribute specifies if a MENU represents an entry point into a hierarchical structure. If a MENU is defined as an entry point, this MENU shall also contain the PURPOSE attribute.

Lexical structure

```
ENTRY (ROOT)<exp>
```

The attribute of ENTRY is specified in Table 72.

Table 72 – ENTRY attribute

Usage	Attribute	Description
m	ROOT	specifies that the MENU may be used as an entry point

9.11.2.4 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.

Lexical structure

```
POST_EDIT_ACTIONS [(reference)<exp>]+, DEFINITION
```

The attributes of POST_EDIT_ACTIONS are specified in Table 73.

Table 73 – POST_EDIT_ACTIONS, PRE_EDIT_ACTIONS, POST_READ_ACTIONS, PRE_READ_ACTIONS, POST_WRITE_ACTIONS, PRE_WRITE_ACTIONS attributes

Usage	Attribute	Description
o	reference	is a reference to a METHOD instance
o	DEFINITION	lexical element (see 9.21.1)

9.11.2.5 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.

Lexical structure

```
POST_READ_ACTIONS [(reference)<exp>]+, DEFINITION
```

The attributes of POST_READ_ACTIONS are specified in Table 73.

9.11.2.6 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.

Lexical structure

```
POST_WRITE_ACTIONS [(reference)<exp>]+, DEFINITION
```

The attributes of POST_WRITE_ACTIONS are specified in Table 73.

9.11.2.7 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.

Lexical structure

```
PRE_EDIT_ACTIONS [(reference)<exp>]+, DEFINITION
```

The attributes of PRE_EDIT_ACTIONS are specified in Table 73.

9.11.2.8 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.

Lexical structure
PRE_READ_ACTIONS [(reference)<exp>]+, DEFINITION

The attributes of PRE_READ_ACTIONS are specified in Table 73.

9.11.2.9 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.

Lexical structure
PRE_WRITE_ACTIONS [(reference)<exp>]+, DEFINITION

The attributes of PRE_WRITE_ACTIONS are specified in Table 73.

9.11.2.10 PURPOSE

Purpose

The PURPOSE attribute defines the application purpose of the MENU. This indicates how the MENU is used, e.g. 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 74.

Table 74 – PURPOSE attribute

Usage	Attribute	Description
s	CATALOG	specifies that the elements of the ITEMS attribute of a MENU instance are references to catalog specific information (e.g. an EDD variable containing the ordering number)
s	DIAGNOSE	specifies that the MENU is used for diagnostic actions
s	LOAD_TO_APPLICATION	specifies that the MENU is used for listing VARIABLEs whose values are sent from the device to an application
s	LOAD_TO_DEVICE	specifies that the MENU is used for listing VARIABLEs whose values are sent to the device
s	MENU	specifies a MENU which contains other MENU items or control items normally presented using horizontal rows
s	PROCESS_VALUE	specifies that the MENU is used to display process values
s	SIMULATION	specifies that the MENU is used for simulations actions
s	string	specifies a user-defined PURPOSE for the MENU
s	TABLE	specifies 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 (e.g. function keys, application menus). TABLE should be used for a vertical data and function representation (e.g. tables).

9.11.2.11 ROLE

Purpose

The ROLE attribute specifies in which role context the MENU is used. If a MENU contains sub-menus, only these sub-menus 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 72.

Table 75 – ROLE attribute

Usage	Attribute	Description
M	string	specifies 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.

9.11.2.12 STYLE

Purpose

The STYLE attribute specifies how the MENU is displayed.

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. If no STYLE attribute is defined, DIALOG is the default.

EXAMPLE A menu may contain a bar graph or a XY-diagram for presenting measured data.

Lexical structure

```
STYLE [(DIALOG, WINDOW, string)<exp>]
```

The attributes of STYLE are specified in Table 76.

Table 76 – STYLE attribute

Usage	Attribute	Description
s	DIALOG	specifies that no other MENU can be activated while this MENU is active. The MENU shall be closed before interactions with other MENUs are possible
s	WINDOW	specifies that other MENUs can be activated while this MENU is active. Interactions with all activated MENUs having WINDOW STYLE attributes are possible
s	string	designates a tool specific MENU style. If a tool cannot interpret the string, this STYLE attribute shall be ignored

9.11.2.13 VALIDITY

Purpose

The VALIDITY attribute specifies whether a MENU is displayed or not. If the VALIDITY attribute is not defined the default value is TRUE.

NOTE VALIDITY is normally expressed using a conditional expression (see 9.23).

Lexical structure

```
VALIDITY [(FALSE, TRUE)<exp>]
```

The attributes of VALIDITY are specified in Table 77.

Table 77 – VALIDITY attributes

Usage	Attribute	Description
s	FALSE	specifies that the MENU is not displayed
s	TRUE	specifies that the MENU is displayed

9.11.3 Sequence diagrams for actions

Figure 44 to Figure 50 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 shown in the sequence.

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

In 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.

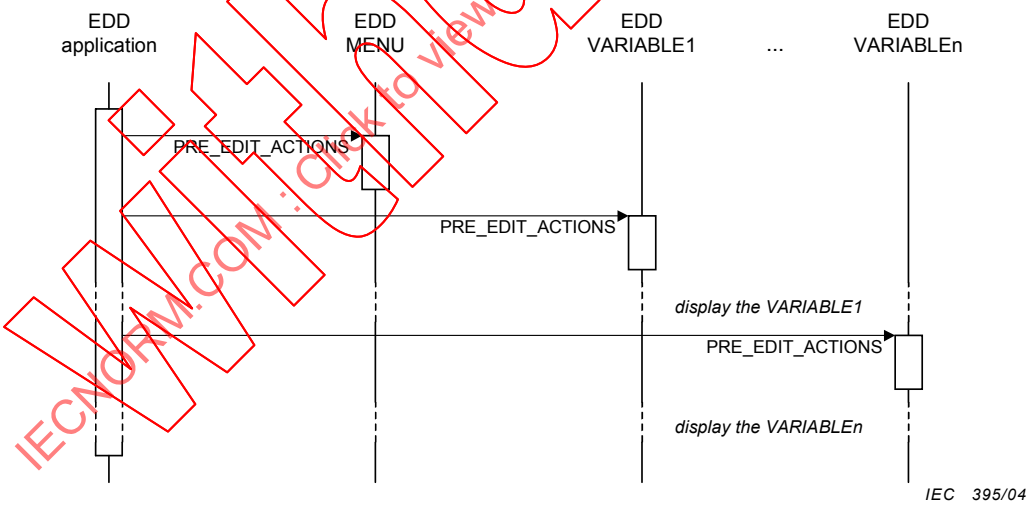


Figure 44 – MENU activation (ACCESS OFFLINE)

Figure 45 – Action performed after a new value is entered

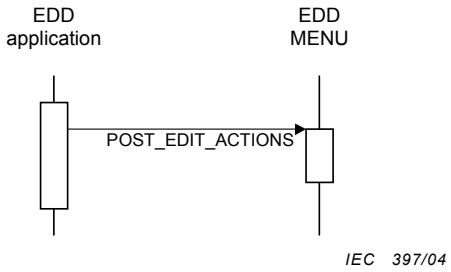


Figure 46 – Action performed after all VARIABLE inputs of the MENU are accepted (ACCESS OFFLINE)

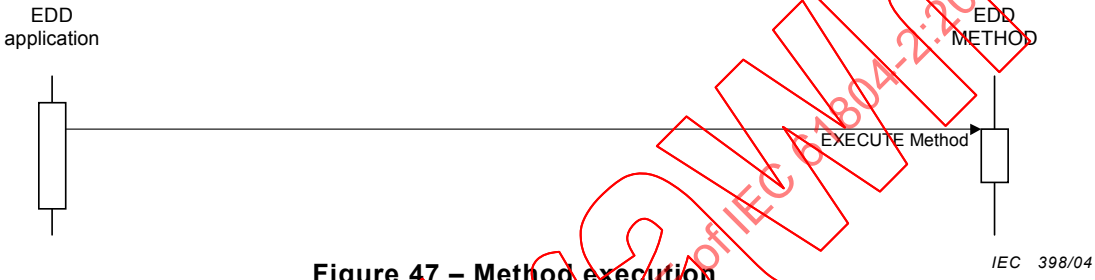


Figure 47 – Method execution

In Figure 47 the case of online access, changed VARIABLE values have to be written to the device before executing the METHOD. After execution of the METHOD, the values are read back from the device and displayed.

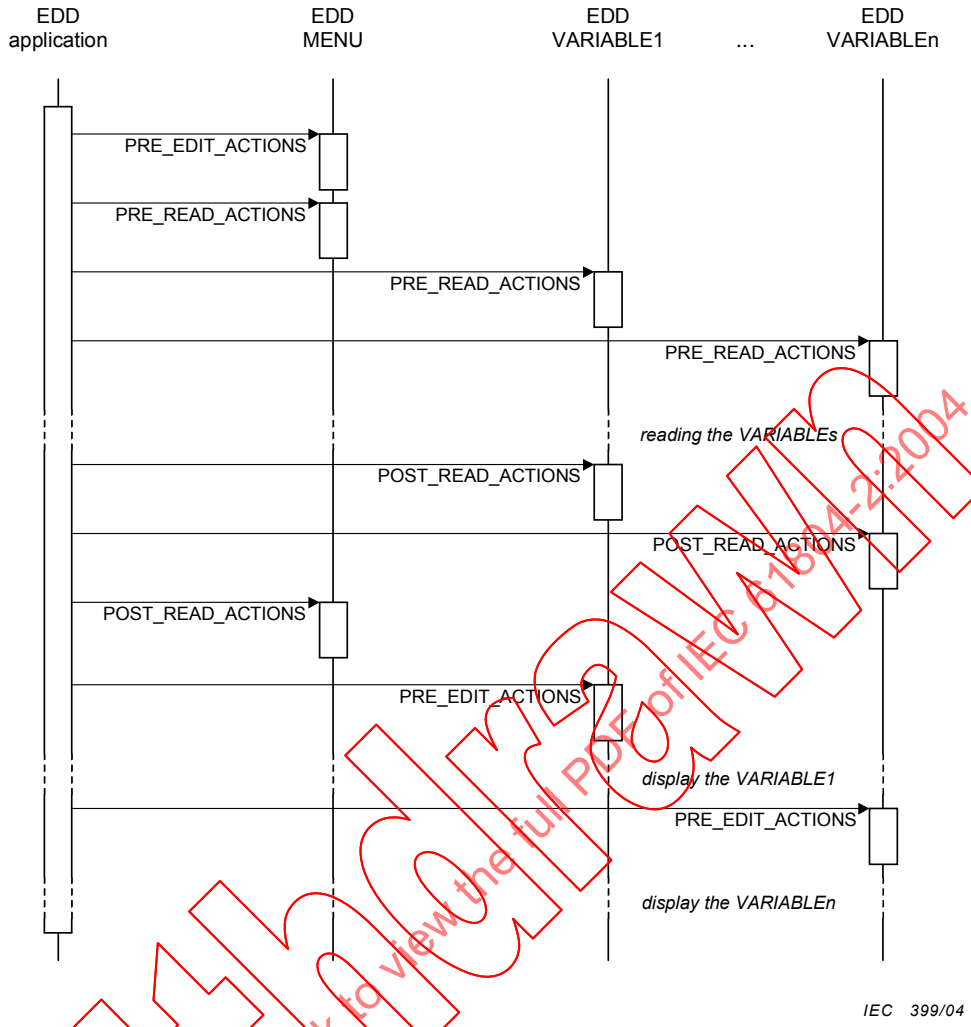


Figure 48 – MENU activation (ACCESS ONLINE)

In Figure 48, the VARIABLEs that have to be read online from a device, the EDD application has to execute appropriate COMMANDs. VARIABLEs may include pre-read/write actions that shall precede the appropriate COMMAND. VARIABLEs post-read/write actions shall be executed following the appropriate COMMAND.

Each VARIABLE or other entity is displayed to the user after its PRE_EDIT_ACTIONS are complete.

PRE_READ_ACTIONS and POST_READ_ACTIONS for different VARIABLEs may be in any convenient order; they shall all be complete before the MENU POST_READ_ACTIONS are executed.

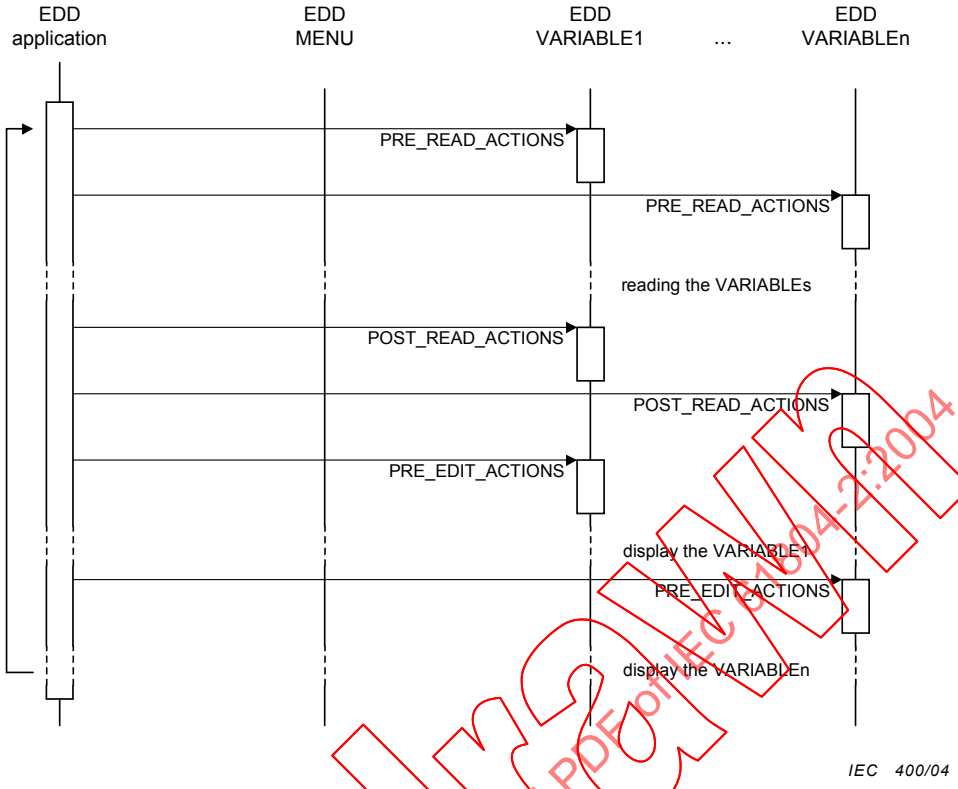


Figure 49 – Cyclic reading of dynamic VARIABLEs (ACCESS ONLINE)

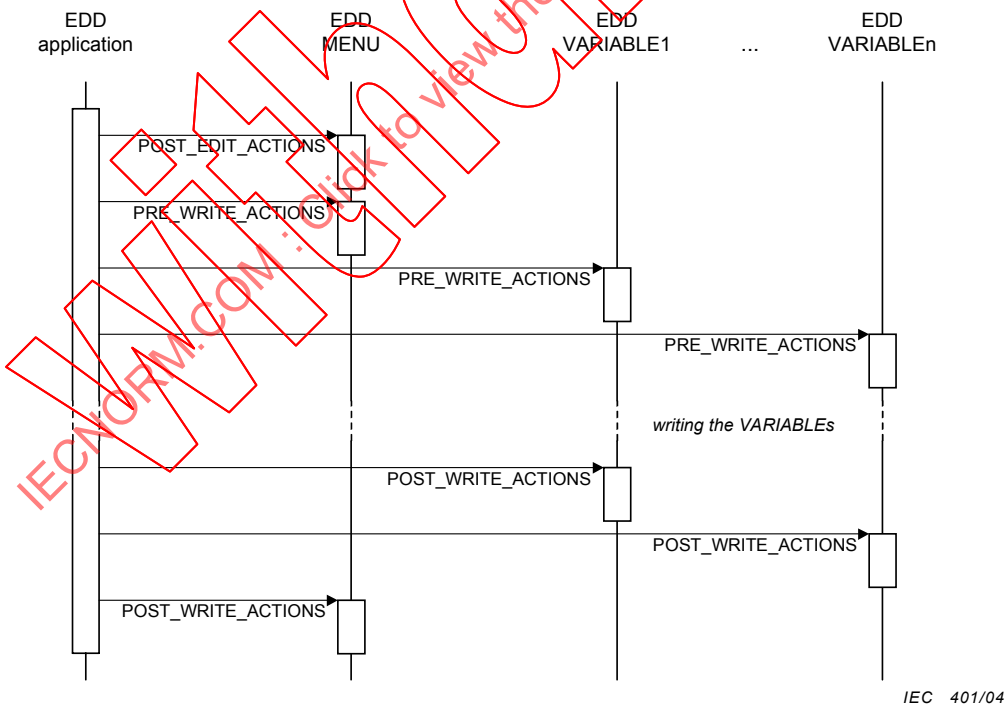


Figure 50 – Action performed after all VARIABLE inputs of the MENU are accepted (ACCESS ONLINE)

In Figure 50, the METHODS are executed after all VARIABLE inputs are accepted. Only the VARIABLE values are written to the device which are changed. After the changes are written, the VARIABLEs shall be read for consistency purposes.

9.12 METHOD

9.12.1 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 [DEFINITION, LABEL, ACCESS, CLASS, HELP, VALIDITY]
```

The attributes of METHOD are specified in Table 78.

Table 78 – METHOD attributes

Usage	Attribute	Description
m	DEFINITION	lexical element (see 9.21.1)
m	LABEL	lexical element (see 9.21.3)
o	ACCESS	lexical element (see 9.12.2.1)
o	CLASS	lexical element (see 9.12.2.2)
o	HELP	lexical element (see 9.21.2)
o	VALIDITY	lexical element (see 9.12.2.3)

9.12.2 Specific attributes

9.12.2.1 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 79.

Table 79 – ACCESS attributes

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

9.12.2.2 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.

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

The attributes of CLASS are specified in Table 94.

9.12.2.3 VALIDITY

Purpose

The VALIDITY attribute specifies whether the METHOD is executed or not. A METHOD without VALIDITY attribute is always valid.

NOTE VALIDITY is normally expressed using a conditional expression (see 9.23).

Lexical structure
VALIDITY [(FALSE, TRUE)<exp>]

The attributes of VALIDITY are specified in Table 80.

Table 80 – VALIDITY attributes

Usage	Attribute	Description
s	FALSE	specifies that the METHOD shall not be executed
s	TRUE	specifies that the METHOD shall be executed

9.13 PROGRAM

9.13.1 General structure

Purpose

PROGRAM allows remote control for 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 [ARGUMENT, RESPONSE_CODES]

The attributes of PROGRAM are specified in Table 81.

Table 81 – PROGRAM attributes

Usage	Attribute	Description
o	ARGUMENT	lexical element (see 9.13.2)
o	RESPONSE_CODES	lexical element (see 9.21.5)

9.13.2 Specific attributes - ARGUMENT

Purpose

ARGUMENTs can be sent to the program during start and resume operations. The PROGRAM arguments are described by the ARGUMENT 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
ARGUMENT [(data-item)<exp>]+

The attribute of ARGUMENT is specified in Table 82.

Table 82 – ARGUMENT attribute

Usage	Attribute	Description
m	data-item	is 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

9.14 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]
```

The attributes of RECORD are specified in Table 83.

Table 83 – RECORD attributes

Usage	Attribute	Description
m	MEMBERS	lexical element (see 9.21.4)
m	LABEL	lexical element (see 9.21.3)
o	HELP	lexical element (see 9.21.2)
o	RESPONSE_CODES	lexical element (see 9.21.5)

9.15 REFERENCE_ARRAY

9.15.1 General structure

Purpose

A REFERENCE_ARRAY is a set of items of the specified item type (e.g. 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]
```

The attributes of REFERENCE_ARRAY are specified in Table 84.

Table 84 – REFERENCE_ARRAY attribute

Usage	Attribute	Description
m	item-type	specifies the type of items listed within the ELEMENTS attribute. It shall be one of the basic elements listed in Table 30
m	ELEMENTS	lexical element (see 9.15.2)
o	HELP	lexical element (see 9.21.2)
o	LABEL	lexical element (see 9.21.3)

9.15.2 Specific attributes - ELEMENTS

Purpose

The ELEMENTS attribute specifies a list of items. It defines for each item the item reference, it's 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 85.

Table 85 – ELEMENTS attribute

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

9.16 Relations

9.16.1 General structure

Purpose

Relations specify relationships between instances of VARIABLEs. The following types of relations are defined:

- REFRESH (see 9.16.2)
- UNIT (see 9.16.3)
- WRITE_AS_ONE (see 9.16.4)

9.16.2 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 86.

Table 86 – REFRESH attributes

Usage	Attribute	Description
m	cause-parameter	specifies a list of VARIABLE instances
m	effected-parameter	specifies 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

9.16.3 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 using that unit code shall be refreshed. When an effected VARIABLE 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 87.

Table 87 – UNIT attributes

Usage	Attribute	Description
m	cause-parameter	is a reference to a VARIABLE instance which contains the unit code
m	effected-parameter	specifies a list of VARIABLE instances using that unit code

9.16.4 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. Not all VARIABLEs sent to the device at the same time are necessarily part of a WRITE_AS_ONE relation. It is up to the EDD application to determine how the updates to the parameters of this relation are passed to the device.

Lexical structure

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

The attribute of WRITE_AS_ONE is specified in Table 88.

Table 88 – WRITE_AS_ONE attribute

Usage	Attribute	Description
m	reference	specifies 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

9.17 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 89.

Table 89 – RESPONSE_CODES attributes

Usage	Attribute	Description
m	description	specifies a string that will be displayed when the RESPONSE_CODE is returned by the device
m	integer	specifies 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
s	SUCCESS	the application layer service was accepted and processed as specified
o	help	specifies help text for the item

9.18 VALUE_ARRAY

9.18.1 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 grouped values.

Lexical structure

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

The attributes of VALUE_ARRAY are specified in Table 90.

Table 90 – VALUE_ARRAY attributes

Usage	Attribute	Description
m	LABEL	lexical element (see 9.21.2)
m	NUMBER_OF_ELEMENTS	lexical element (see 9.18.2.1)
m	TYPE	lexical element (see 9.18.2.2)
o	HELP	lexical element (see 9.21.2)
o	RESPONSE_CODES	lexical element (see 9.21.5)

9.18.2 Specific attributes

9.18.2.1 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 91.

Table 91 – NUMBER_OF_ELEMENTS attribute

Usage	Attribute	Description
s	integer	specifies the number of elements of a VALUE_ARRAY
s	expression	is an expression, which shall be evaluated to a non-negative integer that is within the index range of the NUMBER_OF_ELEMENTS attribute (for the description of expressions, see 9.25.7).

9.18.2.2 TYPE

Purpose

The TYPE attribute is a direct or indirect reference to an EDDL VARIABLE instance. Indirect references to VARIABLE instances shall be specified by following basic constructs:

- BLOCK_A PARAMETERS
- COLLECTIONS
- RECORD
- REFERENCE_ARRAY
- VALUE_ARRAY
- VARIABLE_LIST

whereby the instances of COLLECTIONS and REFERENCE_ARRAY are used for items of type VARIABLE.

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 92.

Table 92 – TYPE attribute

Usage	Attribute	Description
m	reference	is a reference to one of the following EDD basic construct instances: BLOCK_A PARAMETERS, COLLECTIONS, RECORD, REFERENCE_ARRAY, VALUE_ARRAY, VARIABLE_LIST

9.19 VARIABLE

9.19.1 General structure

Purpose

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

Lexical structure
```
VARIABLE identifier, [CLASS, LABEL, TYPE,
HELP, CONSTANT_UNIT, PRE_EDIT_ACTIONS, POST_EDIT_ACTIONS, PRE_READ_ACTIONS,
POST_READ_ACTIONS,  PRE_WRITE_ACTIONS,  POST_WRITE_ACTIONS,  READ_TIMEOUT,
WRITE_TIMEOUT, RESPONSE_CODE, HANDLING, VALIDITY]
```

The attributes of VARIABLE are specified in Table 93.

Table 93 – VARIABLE attributes

Usage	Attribute	Description
m	CLASS	lexical element (see 9.19.2.1)
m	LABEL	lexical element (see 9.21.2)
m	TYPE	lexical element (see 9.19.2.2)
o	CONSTANT_UNIT	lexical element (see 9.19.2.3)
o	HANDLING	lexical element (see 9.19.2.4)
o	HELP	lexical element (see 9.21.2)
o	POST_EDIT_ACTIONS	lexical element (see 9.19.2.5)
o	POST_READ_ACTIONS	lexical element (see 9.19.2.6)
o	POST_WRITE_ACTIONS	lexical element (see 9.19.2.7)
o	PRE_EDIT_ACTIONS	lexical element (see 9.19.2.8)
o	PRE_READ_ACTIONS	lexical element (see 9.19.2.9)
o	PRE_WRITE_ACTIONS	lexical element (see 9.19.2.10)
o	READ_TIMEOUT	lexical element (see 9.19.2.11)
o	RESPONSE_CODE	lexical element (see 9.21.5)
o	STYLE	lexical element (see 9.19.2.12)
o	VALIDITY	lexical element (see 9.19.2.13)
o	WRITE_TIMEOUT	lexical element (see 9.19.2.11)

9.19.2 Specific attributes

9.19.2.1 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 F).

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, TUNE]+
```

The attributes of CLASS are specified in Table 94.

Table 94 – CLASS attributes

Usage	Attribute	Description
s	ALARM	specifies that VARIABLE contains alarm limits
s	ANALOG_INPUT	specifies that VARIABLE is part of an analog input block
s	ANALOG_OUTPUT	specifies that VARIABLE is part of an analog output block
s	COMPUTATION	specifies that VARIABLE is part of a computation block
s	CONTAINED	specifies that VARIABLE represents the physical characteristics of the device
s	CORRECTION	specifies that VARIABLE is part of the correction block
s	DEVICE	specifies that VARIABLE represents the physical characteristics of the device
s	DIAGNOSTIC	specifies that VARIABLE indicates the device status
s	DIGITAL_INPUT	specifies that VARIABLE is part of a digital input block
s	DIGITAL_OUTPUT	specifies that VARIABLE is part of a digital output block
s	DISCRETE_INPUT	specifies that VARIABLE is part of a discrete input block
s	DISCRETE_OUTPUT	specifies that VARIABLE is part of a discrete output block
s	DYNAMIC	specifies that VARIABLE is modified by the device without stimulus from the network
s	FREQUENCY_INPUT	specifies that VARIABLE is part of a frequency input block
s	FREQUENCY_OUTPUT	specifies that VARIABLE is part of a frequency output block
s	HART	specifies that VARIABLE is part of the HART block which characterizes the HART interface. There is only one HART block
s	INPUT	specifies that 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	specifies that 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	specifies that 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	specifies that VARIABLE is used to control a block's operation
s	OUTPUT	specifies that VARIABLE is part of the output block. The values of output VARIABLEs may be accessed by another block input
s	SERVICE	specifies that VARIABLE is used when performing routine maintenance
s	TUNE	specifies that VARIABLE is used to tune the algorithm of a block

9.19.2.2 TYPE

9.19.2.2.1 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 95 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 95.

Table 95 – TYPE attributes

Usage	Attribute	Description	Initial value
s	DOUBLE	arithmetic type (see 9.19.2.2.2.1)	1
s	FLOAT	arithmetic type (see 9.19.2.2.2.1)	1
s	INTEGER	arithmetic type (see 9.19.2.2.2.1)	1
s	UNSIGNED_INTEGER	arithmetic type (see 9.19.2.2.2.1)	1
s	DATE	date/time types (see 9.19.2.2.3)	
s	DATE_AND_TIME	date/time types (see 9.19.2.2.3)	
s	DURATION	date/time types (see 9.19.2.2.3)	
s	TIME	date/time types (see 9.19.2.2.3)	00:00:00
s	TIME_VALUE	date/time types (see 9.19.2.2.3)	
s	BIT_ENUMERATED	enumeration type (see 9.19.2.2.3.1)	0
s	ENUMERATED	enumeration type (see 9.19.2.2.3.2)	the value of the first
s	INDEX	index type (see 9.19.2.2.3.3)	
s	OBJECT_REFERENCE	object references (see 9.19.2.2.4)	
s	ASCII	string type (see 9.19.2.2.5)	
s	BITSTRING	string type (see 9.19.2.2.5)	
s	EUC	string type (see 9.19.2.2.5)	
s	OCTET	string type (see 9.19.2.2.5)	
s	PACKED_ASCII	string type (see 9.19.2.2.5)	
s	PASSWORD	string type (see 9.19.2.2.5)	
s	VISIBLE	string type (see 9.19.2.2.5)	

9.19.2.2.2 Specific attributes

9.19.2.2.2.1 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.

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 96.

Table 96 – DOUBLE, FLOAT, INTEGER, UNSIGNED_INTEGER attributes

Usage	Attribute	Description
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 an INITIAL_VALUE. The DEFAULT_VALUE can also be built by an expression, see 9.26. **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	is a short description of the item
o	DISPLAY_FORMAT	**Purpose** A DISPLAY_FORMAT specifies how the value of the VARIABLE is displayed. The DISPLAY_FORMAT strings are conversion specifiers of the ANSI C (see ISO 9899) functions printf and scanf, respectively. **Lexical structure** DISPLAY_FORMAT (string)<exp>
o	EDIT_FORMAT	**Purpose** A EDIT_FORMAT specifies the format for the editing. The EDIT_FORMAT strings are conversion specifiers of the ANSI C (see ISO 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 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 9.26. **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>
c	m	is an increasing integer starting with one, which identifies the MAX_VALUE if more than one is present

Usage	Attribute	Description
o	MIN_VALUE	**Purpose** The MIN_VALUE 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 9.26. **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	is an increasing integer starting with one, which identifies the MIN_VALUE if more than one is present
o	SCALING_FACTOR	**Purpose** The SCALING_FACTOR 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 9.26. **Lexical structure** SCALING_FACTOR (double)<exp> SCALING_FACTOR (expression)<exp>
o	size	specifies 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	is 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

9.19.2.2.3 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, as follows:

- 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 milliseconds (ms) and an optional day count;
- the data type TIME consists of a time and an optional date;
- the data type TIME_VALUE is used to represent date and time 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 Table F.15.

```
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 Table F.16.

```
DATE_AND_TIME DEFAULT_VALUE, INITIAL_VALUE
```

Lexical structure — DURATION

DURATION specifies that VARIABLE is an integer of 4-6 octets. The first four bits shall have the value zero. Bit 0 to 27 represents the time in the unit ms. The optional number of days is 16 bit binary value. The coding of the octets is specified in Table F.17.

```
DURATION DEFAULT_VALUE, INITIAL_VALUE
```

Lexical structure — TIME

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

The date is stated in days relatively to the 1st January 1984. On the 1st January 1984 the date starts with the value zero. The coding of the octets is specified in Table F.18.

```
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 Table F.19.

```
TIME_VALUE DEFAULT_VALUE, INITIAL_VALUE
```

The attributes of the DATE, DATE_AND_TIME, DURATION, TIME, TIME_VALUE types are specified in Table 96.

9.19.2.2.3.1 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 97.

Table 97 – BIT_ENUMERATED attributes

Usage	Attribute	Description
m	description	see Table 96
m	value	is 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	specifies actions that will be executed when the bit is set
o	function	specifies 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 only the CLASS of the VARIABLE need be specified
o	help	see Table 96
o	size	see Table 96

Usage	Attribute	Description
o	status-class	specifies the meaning of the bit. A status bit may belong to more than one status class (see Table 98). If the VARIABLE is not a status octet, the bits do not have status classes
o	DEFAULT_VALUE	see Table 96
o	INITIAL_VALUE	see Table 96

Table 98 shows the status-classes for a VARIABLE of type BIT_ENUMERATED.

Table 98 – status–class attributes

Usage	Attribute	Description
m	DATA	indicates an invalid data configuration
m	HARDWARE	indicates a hardware failure
m	MISC	indicates a miscellaneous condition
m	MODE	indicates that the device is in a particular mode
m	PROCESS	indicates a problem with the process connected to the field device
m	SOFTWARE	indicates a software failure
m	CORRECTABLE	indicates that the bit can be cleared by the EDD application or the connected process
m	SELF_CORRECTING	indicates that the bit will be reset without further intervention
m	UNCORRECTABLE	indicates that the bit is neither self-correcting nor correctable
m	EVENT	indicates a one-time event
m	STATE	indicates that the device is in a particular state.
m	COMM_ERROR	indicates a communications failure in the other device
o	IGNORE_IN_HANDHELD	indicates that a bit should be ignored in hand-held communicators
o	IGNORE_IN_TEMPORARY_MASTER	indicates that a bit should be ignored in temporary master devices
m	MORE	indicates that there is additional status information available from the device
o	ALL	indicates the impact on all the outputs (see Table 99)
o	AO	indicates the impact on one of the analog outputs (see Table 99)
m	BAD	indicates that the output is unreliable and should not be used for control (see Table 99)
o	DV	indicates an impact on one of the dynamic variables (see Table 99)
o	TV	indicates the impact on one of the transmitter variables (see Table 99)
m	DETAIL	indicates that the bit is summarized elsewhere in a bit of class summary
m	SUMMARY	indicates that the bit is a logical combination of other bits of class detail

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 99.

Table 99 – ALL, AO, DV, TV attributes

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

9.19.2.2.3.2 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 BIT_ENUMERATED type are specified in Table 97.

Table 100 – Enumerated types attributes

Usage	Attribute	Description
m	description	(see Table 96)
m	value	is a constant that specifies the VARIABLE value. The type is 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	(see Table 96)
o	help	(see Table 96)
o	INITIAL_VALUE	(see Table 96)
o	size	(see Table 96)

9.19.2.2.3.3 INDEX

Purpose

A VARIABLE of TYPE INDEX is an unsigned integer, which is used to reference the elements of a REFERENCE_ARRAY (see 9.15). The integer attribute of ELEMENTS (see 9.15.2) 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 101.

Table 101 – Index type attributes

Usage	Attribute	Description
o	size	specifies 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	is a reference to a REFERENCE_ARRAY instance

9.19.2.2.4 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 102.

Table 102 – Object reference type attribute

Usage	Attribute	Description
o	DEFAULT_REFERENCE	specifies that 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 103)

The attributes of the DEFAULT-REFERENCE are specified in Table 103.

Table 103 – 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 last object in the hierarchy
s	NEXT	navigates to the next object of the same hierarchy
s	PARENT	navigates one hierarchy bottom-up
s	PREV	navigates to the previous object of the same hierarchy
s	reference	is a reference to a device projected in the network
s	SELF	navigates back to themselves in their own hierarchy

9.19.2.2.5 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 (e.g. China: EUC-CN, Taiwan EUC-TW). EUC is used to handle East Asian languages (ISO/IEC 10646-1).
- The EDDL processor has to 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 F.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 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 104.

Table 104 – String types attributes

Usage	Attribute	Description
o	size	(see Table 96)
o	DEFAULT_VALUE	(see Table 96)
o	INITIAL_VALUE	(see Table 96)
o	string	is 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 96)
o	help	(see Table 96)

9.19.2.3 CONSTANT_UNIT

Purpose

The CONSTANT_UNIT attribute is used, if a VARIABLE has a units code which never changes. The CONSTANT_UNITS 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 105.

Table 105 – CONSTANT_UNIT attribute

Usage	Attribute	Description
m	string	specifies the units code string to be displayed

9.19.2.4 HANDLING

Purpose

The HANDLING attribute specifies the operations which can be performed on the VARIABLE. By default, a VARIABLE 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 106.

Table 106 – HANDLING attribute

Usage	Attribute	Description
s	READ	specifies that the value of a VARIABLE may be read
s	READ_WRITE	specifies that the value of a VARIABLE may be read and written
s	WRITE	specifies that the value of a VARIABLE may be written

9.19.2.5 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.

Lexical structure
`POST_EDIT_ACTIONS [(reference)<exp>]+, DEFINITION`

The attributes of POST_EDIT_ACTIONS are specified in Table 107.

Table 107 – POST_EDIT_ACTIONS, PRE_EDIT_ACTIONS, POST_READ_ACTIONS, PRE_READ_ACTIONS, POST_WRITE_ACTIONS, PRE_WRITE_ACTIONS attributes

Usage	Attribute	Description
o	reference	is a reference to a METHOD instance
o	DEFINITION	lexical element (see 9.21.1)

9.19.2.6 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.

Lexical structure
```
POST_READ_ACTIONS [(reference)<exp>]+, DEFINITION
```

The attributes of POST_READ_ACTIONS are specified in Table 107.

9.19.2.7 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.

Lexical structure
```
POST_WRITE_ACTIONS [(reference)<exp>]+, DEFINITION
```

The attributes of POST_WRITE_ACTIONS are specified in Table 107.

9.19.2.8 PRE_EDIT_ACTIONS

Purpose

The PRE_EDIT_ACTIONS attribute specifies METHODs that shall be executed immediately when the VARIABLE is displayed. 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.

Lexical structure
```
PRE_EDIT_ACTIONS [(reference)<exp>]+, DEFINITION
```

The attributes of PRE_EDIT_ACTIONS are specified in Table 107.

9.19.2.9 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.

Lexical structure
```
PRE_READ_ACTIONS [(reference)<exp>]+, DEFINITION
```

The attributes of PRE_READ_ACTIONS are specified in Table 107.

9.19.2.10 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.

Lexical structure
```
PRE_WRITE_ACTIONS [(reference)<exp>]+, DEFINITION
```

The attributes of PRE_WRITE_ACTIONS are specified in Table 107.

9.19.2.11 READ_TIMEOUT, WRITE_TIMEOUT

Purpose

A READ_ TIMEOUT specifies the length of time, in ms, the EDD application shall wait for the returned VARIABLE. Similarly, a WRITE_TIMEOUT specifies the length of time, in ms, a EDD application shall wait for the confirmation if 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.

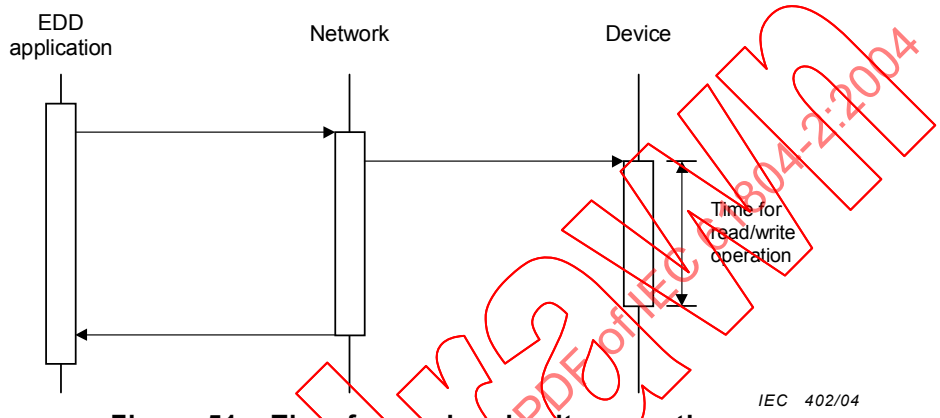


Figure 51 – Time for read and write operation

Lexical structure

```
READ_TIMEOUT (integer)<exp>
WRITE_TIMEOUT (integer)<exp>
```

The attribute of READ_TIMEOUT and WRITE_TIMEOUT is specified in Table 108.

Table 108 – READ/WRITE_TIMEOUT attributes

Usage	Attribute	Description
m	integer	specifies the time period in milliseconds that the EDD application wait for a device to complete its requested update

9.19.2.12 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 109.

Table 109 – STYLE attribute

Usage	Attribute	Description
m	string	specifies an string which is used for display purpose

9.19.2.13 VALIDITY

Purpose

The VALIDITY attribute specifies whether a VARIABLE is valid or invalid. A VARIABLE is invalid only if the VARIABLE does not exist. If the value is simply bad it is still valid. A VARIABLE without a VALIDITY is always valid.

NOTE 1 VALIDITY should be expressed using a conditional (IF, IF-ELSE, SELECT).

NOTE 2 If a VARIABLE is invalid, it should not be displayed.

Lexical structure

```
VALIDITY [(FALSE, TRUE)<exp>]
```

The attributes of VALIDTY are specified in Table 110.

Table 110 – VALIDITY attributes

Usage	Attribute	Description
s	FALSE	specifies that the VARIABLE instance is invalid
s	TRUE	specifies that the VARIABLE instance is valid

9.20 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 111.

Table 111 – VARIABLE_LIST attributes

Usage	Attribute	Description
m	MEMBERS	lexical element (see 9.21.4)
o	HELP	lexical element (see 9.21.2)
o	LABEL	lexical element (see 9.21.3)
o	RESPONSE_CODES	lexical element (see 9.21.5)

9.21 Common attributes

9.21.1 DEFINITION

Purpose

A DEFINITION specifies actions to be performed and is based on a procedural programming language (e.g. ANSI C). In the DEFINITIONs, METHODs and Builtins (see Annex D) may be called. Within a DEFINITION, a EDD basic elements can be used.

The following items shall be supported.

- Basic data types (see F.5.1)
- Arrays (see F.5.1)
- Arithmetic operators (see F.5.1)

- Statements (see F.5.1)
- 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 112.

Table 112 – DEFINITION attributes

Usage	Attribute	Description
o	compound-statement	specifies a block of instructions for a supported procedural programming language

9.21.2 HELP

Purpose

HELP 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 113.

Table 113 – HELP attribute

Usage	Attribute	Description
m	string	specifies user readable text

9.21.3 LABEL

Purpose

LABEL specifies the displayed designation of an element.

Lexical structure
`LABEL (string)<exp>`

The attribute of LABEL is specified in Table 114.

Table 114 – LABEL attribute

Usage	Attribute	Description
m	string	specifies user readable text

9.21.4 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).

Lexical structure
MEMBERS [(identifier, reference, description, help)<exp>]+

The attributes of MEMBERS are specified in

Table 115.

Table 115 – MEMBERS attributes

Usage	Attribute	Description
m	identifier	specifies the name by which the item may be referenced
m	reference	is the reference to an EDD item. The type of the reference depends on the basic constructs If MEMBERS is a COLLECTION attribute, reference is a reference to a BLOCK_A, BLOCK_B, COLLECTION, COMMAND, CONNECTION, DOMAIN, EDIT_DISPLAY, MENU, METHOD, PROGRAM, RECORD, REFERENCE_ARRAY, REFRESH, RESPONSE_CODES, UNIT, VALUE_ARRAY, VARIABLE, VARIABLE_LIST, WRITE_AS_ONE instance If MEMBERS is a RECORDS attribute, reference is a reference to a VARIABLE instance If MEMBERS is a VARIABLE_LISTS attribute, reference is a reference to a RECORDS, VALUE_ARRAY or VARIABLE instance
o	description	provides a short description of the item
o	help	specifies help text for the item

9.21.5 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.

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 116.

Table 116 – RESPONSE_CODES attribute

Usage	Attribute	Description
m	reference	is 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 89.

9.22 Output redirection (OPEN and CLOSE)

Purpose

The output of the EDDL processor can be written to output files. The OPEN keyword instructs the EDDL processor to create a separate output file for the EDD constructs and the CLOSE keyword to close the output file. An attempt to open an output file that is already open results in an error. Opening and closing the same output file more than once in an EDD causes the processor to completely overwrite the contents of the file each time it is opened.

- The OPEN keyword opens an output file in overwrite mode. Each time an OPEN keyword is processed, an output file is opened, and created if necessary, unless the file is already open, in which case an error results. All objects generated are written to all open files.
- The CLOSE keyword prevents further output to an open file. If a file is reopened after closing, then any following objects overwrite the file.

Lexical structure

```
OPEN filename, [(construct)<exp>]+
CLOSE filename
```

The attributes are specified in Table 117.

Table 117 – OPEN and CLOSE attributes

Usage	Attribute	Description
M	filename	is a string of letters or digits which provides the file system name
M	construct	a list of constructs which are written in a separate output file

9.23 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.

NOTE 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 9.25.7). 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 118.

Lexical structure for the SELECT conditional

```
SELECT select-expression, (CASE integer_n, select-clause)+, (DEFAULT,
default-clause)
```

The attributes are specified in Table 118.

Table 118 – IF, SELECT conditional

Usage	Attribute	Description
m	if-expression	is 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	is 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	is 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	is 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	specifies a number which identifies the according CASE. If integer matches with integer_n, the according select_clause is used
m	select-clause	is 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	is 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,
- the 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.

9.24 Referencing

9.24.1 Referencing an EDD instance

Purpose

References are used within an EDD to refer to other EDD items.

EXAMPLE A PRE_EDIT_ACTION of VARIABLE refer to METHOD instances defined elsewhere in the EDD.

Lexical structure

The attribute is specified in Table 119.

Table 119 – Referencing an EDD instance

Usage	Attribute	Description
m	identifier	is the identifier of an EDD instance

9.24.2 Referencing members of a RECORD

Purpose

This referencing is used to access a member of a RECORD instance.

Lexical structure
`identifier, member-identifier`

The attributes are specified in Table 120.

Table 120 – Referencing elements of RECORD

Usage	Attribute	Description
m	identifier	is the identifier of a RECORD instance
m	member-identifier	is the identifier of an item of MEMBERS

9.24.3 Referencing elements of a VALUE_ARRAY

Purpose

This referencing is used to access an element of a VALUE_ARRAY instance.

Lexical structure
`identifier, expression`

The attributes are specified in Table 121.

Table 121 – Referencing elements of VALUE_ARRAY

Usage	Attribute	Description
m	identifier	is the identifier of a VALUE_ARRAY instance
m	expression	is an expression, which shall be evaluated to a non-negative integer that is within the index range of the VALUE_ARRAY (for the description of expressions, see 9.25.7)

9.24.4 Referencing members of a COLLECTION

Purpose

This referencing is used to access a member of a COLLECTION instance.

Lexical structure
`identifier, member-identifier`

Lexical structure if member is a RECORD
`identifier, record-member, member-identifier`

Lexical structure if member is a VALUE_ARRAY
`identifier, value-array-member, expression`

The attributes are specified in Table 122.

Table 122 – Referencing members of COLLECTION

Usage	Attribute	Description
m	identifier	is an identifier of a COLLECTION instance or a COLLECTION reference
m	member-identifier	is the identifier of an item of MEMBERS
m	record-member	is the identifier of an item of MEMBERS. The item references a RECORD instance
m	value-array-member	is the identifier of an item of MEMBERS. The item references a VALUE_ARRAY instance
m	expression	is an expression, which shall be evaluated to a non-negative integer that is within the index range of the VALUE_ARRAY (for the description of expressions, see 9.25.7)

9.24.5 Referencing elements of a REFERENCE_ARRAY

Purpose

This referencing is used to access an element of a REFERENCE_ARRAY instance.

Lexical structure
`identifier, expression1`

Lexical structure if member is a RECORD
`identifier, expression1, member-identifier`

Lexical structure if member is a VALUE_ARRAY
`identifier, expression1, expression2`

The attributes are specified in Table 123.

Table 123 – Referencing members of REFERENCE_ARRAY

Usage	Attribute	Description
m	identifier	is an identifier of a REFERENCE_ARRAY instance or a REFERENCE_ARRAY reference
m	expression1	is 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 9.25.7)
m	member-identifier	is the identifier of an item of MEMBERS
m	expression2	is an expression, which shall be evaluated to a non-negative integer that is within the index range of the VALUE_ARRAY (for the description of expressions, see 9.25.7)

9.24.6 Referencing members of a VARIABLE_LISTS

Purpose

This referencing is used to access a member of a VARIABLE_LISTS instance.

Lexical structure
`identifier, member-identifier`

Lexical structure if member is a RECORD
`identifier, record-member, member-identifier`

Lexical structure if member is a VALUE_ARRAY
`identifier, value-array-member, expression`

The attributes are specified in Table 124.

Table 124 – Referencing members of VARIABLE_LISTS

Usage	Attribute	Description
m	identifier	is an identifier of a VARIABLE_LISTS instance or a VARIABLE_LISTS reference
m	member-identifier	is the identifier of an item of MEMBERS
m	record-member	is the identifier of an item of MEMBERS. The item references a RECORD instance
m	value-array-member	is the identifier of an item of MEMBERS. The item references a VALUE_ARRAY instance
m	expression	is an expression, which shall be evaluated to a non-negative integer that is within the index range of the VALUE_ARRAY (for the description of expressions, see 9.25.7)

9.24.7 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`

Lexical structure if member is a RECORD
`PARAM, record-member, member-identifier`

Lexical structure if member is a VALUE_ARRAY
`PARAM, value-array-member, expression`

The attributes are specified in Table 125.

Table 125 – Referencing members of a BLOCK_A PARAMETERS

Usage	Attribute	Description
m	member-identifier	is the identifier of an item of MEMBERS
m	record-member	is the identifier of an item of MEMBERS. The item references a RECORD instance
m	value-array-member	is the identifier of an item of MEMBERS. The item references a VALUE_ARRAY instance
m	expression	is an expression, which shall be evaluated to a non-negative integer that is within the index range of the VALUE_ARRAY (for the description of expressions, see 9.25.7)

9.24.8 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`

Lexical structure if member is a RECORD
`PARAM_LISTS, record-member, member-identifier`

Lexical structure if member is a VALUE_ARRAY
`PARAM_LISTS, value-array-member, expression`

The attributes are specified in Table 126.

Table 126 – Referencing members of BLOCK_A PARAMETER_LISTS

Usage	Attribute	Description
m	member-identifier	is the identifier of an item of MEMBERS
m	record-member	is the identifier of an item of MEMBERS. The item references a RECORD instance
m	value-array-member	is the identifier of an item of MEMBERS. The item references a VALUE_ARRAY instance
m	expression	is an expression, which shall be evaluated to a non-negative integer that is within the index range of the VALUE_ARRAY (for the description of expressions, see 9.25.7)

9.24.9 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 us specified in Table 127.

Table 127 – Referencing BLOCK_A CHARACTERISTICS

Usage	Attribute	Description
m	characteristics-identifier	is the identifier of the CHARCTERISTICS reference

9.25 Strings

9.25.1 Specifying a string as a string literal

Purpose

A text string can be specified as a string literal (see C.2.3).

Lexical structure
`string`

The attribute is specified in Table 128.

Table 128 – String as a string literal

Usage	Attribute	Description
m	string	specifies a string literal

9.25.2 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 attribute is specified in Table 129.

Table 129 – String as a string variable

Usage	Attribute	Description
m	reference	is a reference to a VARIABLE instance

9.25.3 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 130.

Table 130 – String as an enumeration value

Usage	Attribute	Description
m	reference	is a reference to a VARIABLE instance of TYPE ENUMERATION
m	value	is 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)

9.25.4 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 131.

Table 131 – String as a dictionary reference

Usage	Attribute	Description
m	reference	is a reference to one or more strings specified in a dictionary (see 9.27)

9.25.5 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 132.

Table 132 – Referencing HELP and LABEL attributes of EDD instances

Usage	Attribute	Description
m	reference	is a reference to an EDD instance which contains the attributes LABEL and/or HELP
s	HELP	supplies the HELP string of the instance
s	LABEL	supplies the LABEL string of the instance

9.25.6 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 through 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 133.

Table 133 – String operation

Usage	Attribute	Description
m	string	is either string literals, string variables, string enumeration values, dictionary references or HELP, LABEL references
o	concatenation-operator	specifies that the strings shall be concatenated

9.25.7 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 134.

Table 134 – Format specifier

Usage	Attribute	Description
m	format	consists of the ANSI C (see ISO 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	specifies the LABEL of the VARIABLE instance
o/s	reference	specifies the value or label of the referenced VARIABLE instance
o/s	id	specifies the item IDs of the VARIABLEs

9.26 Expression

9.26.1 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.

9.26.2 Primary expressions

Table 135 below summarizes the primary expressions in the EDDL.

Table 135 – Primary expressions

Primary Expression	Description
constant	is an expression whose value is identical to the value of the constant
parenthesized expression	is an expression whose value is identical to the value of the enclosed expression
variable reference	is an expression whose value is the value of the referenced VARIABLE
ARRAY_INDEX	is an expression whose value was taken from the index number of the referencing VALUE_ARRAY
attribute values of VARIABLEs	is a reference consisting of the identifier of a VARIABLE instance and an attribute. The reference supplies the according value of the attribute (see Table 136 for the returned values)
CURRENT_ROLE	specifies 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

Table 136 – Attribute values of VARIABLEs

Attributes	Data type	Description
MIN_VALUE	type of VARIABLE instance	returns 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 96) shall be used for referencing
MAX_VALUE	type of VARIABLE instance	returns 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 96) shall be used for referencing
SCALING_FACTOR	double	returns a value which complies with the SCALING_FACTOR of the VARIABLE instance
DEFAULT_VALUE	type of VARIABLE instance	returns a value which complies with the DEFAULT_VALUE of the VARIABLE instance
INITIAL_VALUE	type of VARIABLE instance	returns a value which complies with the INITIAL_VALUE of the VARIABLE instance
VARIABLE_STATUS		returns a value which indicates the status of VARIABLE. 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, e.g. because the units are not compatible

9.26.3 Unary expressions

A unary expression consists of a numerical operand, an expression, preceded by a unary operator. Table 137 below specifies the unary expressions in the EDDL.

Table 137 – Unary expressions

Operator	Description
++	increments its operand by adding the value 1 to the value of the operand. The increment operator can be used either as prefix operator or as postfix operator. Both the prefix and the postfix notation will cause an increment of the operand. The difference is that the prefix expression changes the operand before using its value, while the postfix expression changes the operand after using its value. Thus the result of these expression can have different meanings according to the program context
--	decrements its operand by subtracting the value 1 from the operand. The decrement operator can be used either as prefix operator or as postfix operator. Both the prefix and the postfix notation will cause a decrement of the operand. The difference is that the prefix expression changes the operand before using its value, while the postfix expression changes the operand after using its value. Thus the result of these expression can have different meanings according to the program context
-	is the arithmetic negation of its operand
~	is 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
!	is the logical negation of its operand

9.26.4 Binary expressions

9.26.4.1 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
- Assignments

9.26.4.2 Multiplicative operators

Multiplicative operators specify multiplication and division of numerical operands. Table 138 specifies the multiplicative operators.

Table 138 – Multiplicative operators

Operator	Description
*	specifies the multiplication of its operands
/	specifies the division of the first operand by the second operand. The result of the / operator is the quotient of the division
%	specifies the division of the first operand by the second operand. The result of the % operator is the remainder

9.26.4.3 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 139 specifies additive operators.

Table 139 – Additive operators

Operator	Description
+	specifies the addition of its operands
-	specifies the subtraction of the second operand from the first

9.26.4.4 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 140 specifies the shift operators.

Table 140 – Shift operators

Operator	Description
<<	shifts the first operand to the left. The bits shifted off are discarded, and the vacated bits are zero filled
>>	shifts 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

9.26.4.5 Relational operators

Relational operators (<, <=, >, >=) specify a comparison of its operands. The result of this type of an expression is 1 if the tested relationship is true, otherwise the result is 0. The operands shall be numerical data types. Table 141 specifies the relational operators.

Table 141 – 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“

9.26.4.6 Equality operators

The equality operators are == and !=. The result of this type of an 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 142 specifies the equality operators.

Table 142 – Equality operators

Operator	Description
==	tests for the relationship “equals“
!=	tests for the relationship “does not equal“

9.26.4.7 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.

9.26.4.8 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.

9.26.4.9 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.

9.26.4.10 Logical AND operator (&&)

The && operator specifies the Boolean AND evaluation of its integer operands. 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.

9.26.4.11 Logical OR operator (||)

The || operator specifies the Boolean OR evaluation of its integer operands. 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.

9.26.4.12 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 ;
```

9.26.4.13 Assignments

Procedural programming languages provide a series of assignment operators, all of which are right-associative. All assignment operators require an unary expression as their left operand. The right operand can be an assignment expression again. The type of the assignment expression corresponds to its left operand. The value of an assignment operation is the value stored in the left operand after the assignment has taken place.

The binary = operator indicates the simple assignment. The binary operators *=, /=, %=, +=, -=, >>=, <<=, &=, ^=, and |= indicate a compound assignment expression.

A compound assignment of general form as in

```
expr1 <operator>= expr2
```

is equivalent with the expression

```
expr1 = expr1 <operator> (expr2)
```

The data type of expr2 shall be convertible to the data type of expr1. The semantic meaning of the assignments are specified in 9.26.2, 9.26.3 and 9.26.4.

9.26.4.14 Expression list

Each expression can consist of a list of comma-separated binary expressions. A pair of expressions separated by a comma is evaluated left-to-right and the value of the left expression is discarded. The type and value of the result are the type and value of the right operand.

EXAMPLE 1 c -= (a=5, b=3-a, c=b+a, a*=b+=4+2*a);
where the values 60, 12, and -57 are stored to the variables a, b, and c, respectively.
EXAMPLE 2 fct (x, (y=8, y*25.4), z)
has three arguments, the second of which has the value 203.2. In contexts where the comma is given a special meaning, e.g., in a list of actual function parameters and lists of initializers, the comma operator can only appear in parentheses; e.g., the function call

9.27 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 (e.g., 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, string)+
```

The attributes are specified in Table 143.

Table 143 – Text dictionary attributes

Usage	Attribute	Description
m	identifier	is used to reference the text string in an EDD
m	language-code	specifies the language used and the ASCII or multi code presentation of a string
m	string	specifies the text string in the related language

EXAMPLE

write_protected
"Write Protected"
"|de|Schreibgeschützt"
"|fr|Protégé en écriture"
"|it|Protetto in scrittura"
"|es|Protegido contra escritura"
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. To support more than these languages, the qualifier could be replaced, e.g. by a language code used for telephone numbers or based on ISO 639.

10 Conformance statement

A conformant device shall implement all the mandatory requirements of this standard (recognizable by the “shall …” declaration). The manufacturer of a device shall declare conformance of the device to IEC 61804-2. To show the optional features, the conformance declaration shall use the template conventions given in Annex B.

The manufacturer of a device shall declare conformance of the EDDL profile used (see Annex F). If a manufacturer or a group of manufacturers is planning to develop a new selection list (profile) or change an existing selection list, a new profile should be added to the annex of this standard. This is also necessary if a new syntax is specified and used in conjunction with the EDDL lexical structure.

Annex A
(informative)

Parameter description

Annex A describes the parameter of the different FBs, see Table A.1.

NOTE the acronyms have the meaning: M=mandatory, O=optional, C=conditional, R=read, R/W=read/write.

Table A.1 – Parameter description

Parameter Name	Description	Data type	User Access Read /Write	Class M/O/C
Analog Input FB				
MEASUREMENT_VALUE	Main measurement value as a result of the Measurement FB	Numeric	R	M
MEASUREMENT_STATUS	Status of the MEASUREMENT_VALUE	List of Boolean	R	M
PRIMARY_MEASUREMENT _VALUE	Primary measurement value as a result of the measurement technology block	Numeric	R	M
PRIMARY_MEASUREMENT _STATUS	Status of the PRIMARY_MEASUREMENT _VALUEparameter	List of Boolean	R	M
UNITS	Units of the main measurement value	Enumerated	R/W	O
HIGH_ALARM_LIMIT	Value for upper limit of alarms	Numeric	R/W	O
LOW_ALARM_LIMIT	Value for lower limit of alarms	Numeric	R/W	O
MODE	Operation mode of the block (e.g. Manual, Automatic, Remote Cascade)	Enumerated	R/W	O
CHANNEL	Logical reference to the technology block measurement	Enumerated	R/W	O
SIMULATE	Used to carry out internal tests	Enumerated	R/W	O
Analog Output FB				
REMOTE_SETPOINT_VALUE	Remote Setpoint from the output of an upstream application block	Numeric	R/W	M
REMOTE_SETPOINT_STATUS	Status of the REMOTE_SETPOINT_VALUE parameter	List of Boolean	R/W	M
OUT_VALUE	Primary output value of the analog actuation output function	Numeric	R/W	M
OUT_STATUS	Status of the OUT_VALUE parameter	List of Boolean	R/W	M
READBACK_VALUE	Feedback of the downstream technology block readback output value	Numeric	R/W	M
READBACK_STATUS	Status of the READBACK_VALUE parameter	List of Boolean	R	M
READBACK_OUT_VALUE	Feedback to the upstream application block readback value	Numeric	R/W	M
READBACK_OUT_STATUS	Status of the READBACK_OUT_VALUE parameter	List of Boolean	R/W	M
UNITS	Unit selection	Enumerated	R/W	O
SP_HI_LIM	Setpoint value high limit	Numeric	R/W	O
SP_LO_LIM	Setpoint value low limit	Numeric	R/W	O
MODE	Operation mode of the block (e.g. Manual, Automatic, Remote Cascade)	Enumerated	R/W	O
CHANNEL	Reference to the technology block actuator	Enumerated	R/W	O

Parameter Name	Description	Data type	User Access Read /Write	Class M/O/C
SIMULATE	Used to carry out internal tests	Enumerated	R/W	O
Discrete Input FB				
DISC_MEASUREMENT_VALUE	Discrete input measurement value	Boolean	R	M
DISC_MEASUREMENT _STATUS	Status of the DISC_MEASUREMENT_VALUE parameter	List of Boolean	R	M
DISC_PRIMARY _MEASUREMENT_VALUE	Primary discrete measurement value as a result of the discrete input technology block	Boolean	R	M
DISC PRIMARY_MEASUREMENT _STATUS	Status of the DISC_PRIMARY _MEASUREMENT_VALUE parameter	List of Boolean	R	M
CONVERT	Boolean invert of the discrete primary value or of the sensor value	Boolean	R/W	O
MODE	Operation mode of the block, (e.g. Manual, Automatic, Remote Cascade)	Enumerated	R/W	O
CHANNEL	Reference to the technology block input	Enumerated	R/W	O
SIMULATE	Used to carry out internal tests	Enumerated	R/W	O
Discrete Output FB				
DISC_REMOTE_SETPOINT_VA LUE	Discrete Remote Setpoint from the output of an upstream application block	Boolean	R/W	M
DISC_REMOTE_SETPOINT _STATUS	Status of DISC_REMOTE_SETPOINT_VALUE parameter	List of Boolean	R/W	M
DISC_OUT_VALUE	Primary output value of the on/off actuation output function	Numeric	R/W	M
DISC_OUT_STATUS	Status of the DISC_OUT_VALUE parameter	List of Boolean	R	M
DISC_READBACK_VALUE	Readback of the discrete readback output from a downstream technology block	Boolean	R/W	M
DISC_READBACK_STATUS	Status of the DISC_READBACK_VALUE parameter	List of Boolean	R/W	M
DISC_READBACK_OUT_VALUE	Feedback to the upstream application block discrete readback value	Numeric	R/W	M
DISC_READBACK_OUT _STATUS	Status of the DISC_READBACK_OUT_VALUE parameter	List of Boolean	R/W	M
MODE	Operation mode of the block, (e.g. Manual, Automatic, Remote Cascade)	Enumerated	R/W	O
CHANNEL	Reference to the technology block of the actuator	Enumerated	R/W	O
SIMULATE	Used to carry out internal tests of the actuator	Enumerated	R/W	O
Calculation FB				
FOLLOW	Forces the Output value to track a block input	Numeric	R/W	O
IN_VALUE	Primary input value to the calculation	Numeric	R	M
IN_STATUS	Status of the primary input value	List of Boolean	R	M
OUT_VALUE	Primary output value of the calculation	Numeric	R/W	M
OUT_STATUS	Status of the primary output value	List of Boolean	R	M
READBACK_VALUE	Feedback of the downstream block readback output value	Numeric	R/W	M

Parameter Name	Description	Data type	User Access Read /Write	Class M/O/C
READBACK_STATUS	Status of the readback value	List of Boolean	R/W	M
READBACK_OUT_VALUE	Feedback to the upstream block readback value	Numeric	R/W	M
READBACK_OUT_STATUS	Status of the readback output value	List of Boolean	R/W	M
Control FB				
IN_VALUE	Primary input measurement	Numeric	R	M
IN_STATUS	Status of primary input measurement	List of Boolean	R	M
OUT_VALUE	Primary output value of the control function	Numeric	R/W	M
OUT_STATUS	Status of the OUT_VALUE parameter	List of Boolean	R	M
READBACK_VALUE	Feedback of the downstream block readback output value	Numeric	R/W	M
READBACK_STATUS	Status of the READBACK_VALUE parameter	Numeric	R/W	M
READBACK_OUT_VALUE	Feedback to the upstream block readback value	Numeric	R/W	M
READBACK_OUT_STATUS	Status of the READBACK_OUT_VALUE parameter	Numeric	R/W	M
REMOTE_SETPOINT_VALUE	Remote target value for a process output measurement from an upstream application block	Numeric	R/W	M
REMOTE_SETPOINT_STATUS	Status of the REMOTE_SETPOINT_VALUE parameter	List of Boolean	R	M
SETPOINT	Local target value for a process output measurement	Numeric	R/W	M
SP_HI_LIM	Upper limit for Setpoint value	Numeric	R/W	O
SP_LO_LIM	Lower limit for Setpoint value	Numeric	R/W	O
ALARM_HI	Upper alarm limit for the primary input value	Numeric	R/W	O
ALARM_LO	Lower alarm limit for the primary input value	Numeric	R/W	O
MODE	Operation mode of the block, (e.g. Manual, Automatic, Remote Cascade)	Enumerated	R/W	O
Temperature Technology Block				
RAW_MEASUREMENT_VALUE	Raw measurement value as result of measurement acquisition	Numeric	R	M
RAW_MEASUREMENT_STATUS	Status of RAW_MEASUREMENT_VALUE parameter	List of Boolean	R	M
PRIMARY_MEASUREMENT _VALUE	Primary measurement value as result of the transformation function	Numeric	R	M
PRIMARY_MEASUREMENT _STATUS	Status of PRIMARY_MEASUREMENT _VALUE parameter	List of Boolean	R	M
SECONDARY_MEASUREMENT _VALUE	Secondary measurement value(s) as result of the transformation function	Numeric	R	O
SECONDARY_MEASUREMENT _STATUS	Status of the corresponding SECONDARY_MEASUREMENT_VALUE parameters	List of Boolean	R	O
CHANGE_CONFIG	Wiring check	Enumerated	R	O
SENSOR_CONNECTION	2, 3 or 4 wires for RTD measurement	Enumerated	R/W	O

Parameter Name	Description	Data type	User Access Read /Write	Class M/O/C
SENSOR_TYPE	Thermocouple, Thermoresistance (RTD), Low voltage i.e. in the range +/-25 mV or +/-100 mV	Enumerated	R/W	M
AD_CONV	A/D Conversion Parameters	Numeric	R/W	O
TEST_COMMAND	Starts test procedure to check the sensor	Enumerated	R/W	O
COMPENS_PARAM	Cold junction Compensation Parameters	Numeric	R/W	O
LINE_TYPE	Linearization curve coefficients, Supplementary Measure parameters	Enumerated	R/W	O
FILTER_PARAM	Filter parameters, (e.g. Anti-aliasing pre-filtering)	Enumerated	R/W	O
Pressure Technology Block				
RAW_MEASUREMENT_VALUE	Raw measurement value as result of measurement acquisition	Numeric	R	M
RAW_MEASUREMENT_STATUS	Status of RAW_MEASUREMENT_VALUE parameter	List of Boolean	R	M
PRIMARY_MEASUREMENT _VALUE	Primary measurement value as result of the transformation function	Numeric	R	M
PRIMARY_MEASUREMENT _STATUS	Status of PRIMARY_MEASUREMENT _VALUE parameter	List of Boolean	R	M
SECONDARY_MEASUREMENT _VALUE	Secondary measurement value(s) as result of the transformation function	Numeric	R	O
SECONDARY_MEASUREMENT _STATUS(es)	Status of SECONDARY_MEASUREMENT_VALUE parameters	List of Boolean	R	O
SENSOR-CODE	Type of Sensor (it identifies the transformation curve to be used)	Enumerated	R/W	O
CAL_POINT_LO	This parameter contains the lowest calibrated value, which is put to the sensor and transfer this point as LOW to the transmitter.	Numeric	R/W	O
CAL_POINT_HI	This parameter contains the highest calibrated value, which is put to the sensor and transfert this point as HIGH to the transmitter.	Numeric	R/W	O
SENSOR_HI_LIM	Physical upper limit of the sensor	Numeric	R/W	O
SENSOR_LO_LIM	Physical lower limit of the sensor	Numeric	R/W	O
TEST_COMMAND	Starts test procedure to check the sensor	Enumerated	R/W	O
TRANSF_PARAM	Linearization curve coefficients and supplementary measure parameters	Numeric	R/W	O
LOW_FLOW_CUT_OFF	Lowest flow value which is determined as the minimum value	Numeric	R/W	O
FILTER_PARAM	Filter parameters, (e.g. Anti-aliasing pre-filtering)	Enumerated	R/W	O
Modulating Actuation Technology Block				
SETPOINT_VALUE	Setpoint value for a process output from an upstream application block	Numeric	R/W	M
SETPOINT_STATUS	Status of the SETPOINT_STATUS parameter	List of Boolean	R/W	M
READBACK_VALUE	Feedback to the upstream AB readback value	Numeric	R	M

Parameter Name	Description	Data type	User Access Read /Write	Class M/O/C
READBACK_STATUS	Status of the READBACK_VALUE parameter	List of Boolean	R	M
ACTUATOR_DEMAND	Demand to the actuator resulting from the transformation function	Enumerated	R	O
POSITION_MEASURE	Result feedback from the actuation/acquisition function	Numeric	R	O
FAILSAFE_ACTION	Fail-Safe position for power-loss of the Actuator respectively the valve	Enumerated	R/W	O
TEST_COMMAND	Starts test procedure to check the actuator	Enumerated	R/W	O
SETP_CUTOFF_MIN	When the setpoint (OUT_VALUE) goes below the defined per cent of span, the actuator signal goes to the minimum limit.	Numeric	R/W	O
SETP_CUTOFF_MAX	When the setpoint (OUT_VALUE) goes over the defined per cent of span, the actuator signal goes to the maximum limit.	Numeric	R/W	O
DEADBAND	Deadband of the actuator	Numeric	R/W	O
SELF_CALIB_STATUS:	Result of the calibration procedure (undetermined, aborted, success)	List of Boolean	R	O
On/Off Actuation Technology Block				
DISC_SETPOINT_VALUE	Local target value for the discrete actuation output	Boolean	R/W	M
DISC_SETPOINT_STATUS	Status of the discrete setpoint	List of Boolean	R/W	M
DISC_READBACK_VALUE	Feedback to the upstream application block readback value	Boolean	R	M
DISC_ READBACK_STATUS	Status of the discrete readback output value	List of Boolean	R	M
DISC_ACTUATOR_DEMAND	Demand to the actuator resulting from the transformation function	Boolean	R	O
DISC_POSITION_MEASURE	Result feedback from the actuation/acquisition function	Boolean	R	O
FAILSAFE_ACTION	Fail-Safe position for power-loss of the Actuator respectively the valve	Enumerated	R/W	O
TRAVEL_COUNT	Number of cycles from OPEN to CLOSE and CLOSE to OPEN	Numeric	R	O
TRAVEL_COUNT_LIMIT	Limit for TRAVEL_COUNT	Numeric	R/W	O
BREAK_TIME_CLOSE	Dead time between the change of the state (DISC_SETPOINT_VALUE) from CLOSE and the indication that the actuator starts its action	Numeric	R/W	O
BREAK_TIME_OPEN	Dead time between the change of the state (DISC_SETPOINT_VALUE) from OPEN and the indication that the actuator starts its action	Numeric	R/W	O
SELF_CALIB_STATUS	Result of the calibration procedure (undetermined, aborted, success)	List of Boolean	R	O
Device Block				
DEVICE_VENDOR	Company name of the manufacturer	String	R	M
DEVICE_MODEL	Name of the device model	String	R	M
DEVICE_REVISION	Device revision number	String	R	M
DEVICE_SER_NO	Serial number of the device	String	R	O
DEVICE_STATUS	Status of the device	List of Boolean	R	M

Annex B
(normative)

IEC 61804 Conformance Declaration

The following conventions are given as a guideline and template and are common to all conformance declarations.

The conformance is described as follows. The (sub)clause selection is defined in Table B.1 and Table B.2. The selected options are indicated by (sub)clause and key words. Selection is made at the highest (sub)clause level.

Table B.1 – Conformance (sub)clause selection table

Clause #	Key Word	Presence	Constraints

Table B.2 – Contents of (sub)clause selection tables

Column	Text	Meaning
Clause #	<#>	(sub)clause number of the base specifications
Keyword	<text>	(sub)clause title of the base specifications
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)
	—	presence is defined in the following subclauses
	Partial	parts of this (sub)clause are included in the profile
Constraints	see <#>	constraints/remarks are defined in the given subclause, table or figure of this conformance document
	—	no constraints other than those 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

Annex C
(normative)

EDDL Formal Definition

C.1 EDDL Preprocessor

C.1.1 General structure

Purpose

The preprocessor generates a EDD before compilation. The preprocessor replaces parts of the EDD text, inserts the contents of other files or suppresses 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.

C.1.2 Directives

C.1.2.1 #define

Purpose

The #define directive give 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 identifier in the EDD file with token-string.

NOTE 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 a 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. The following rules shall apply:

- 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 space between the name and the “(“ is allowed.

C.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.

NOTE No additional tokens are permitted on the directive line after the final “ or >.

C.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.

C.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 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 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.

C.1.2.5 #ifdef, #ifndef and #undef

Purpose

The #ifdef and #ifndef directives controls 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.

C.1.3 Predefined macros

C.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.

C.1.3.2 List of predefined macros

C.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.

C.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.

C.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.

C.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 preceded by two forward slashes (//) is supported. A comment beginning with two forward slashes is terminated by the next NEWLINE character that is not preceded by an escape character.

C.2 Conventions

C.2.1 Integer constants

An integer constant may be specified in binary, octal, decimal, or hexadecimal notation. Table C.1 specifies the conventions for each type of notation.

Table C.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

C.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.

C.2.3 String literals

A string literal is a possibly empty sequence of characters enclosed in double quotes ("). The enclosed characters shall be any ISO Latin–1 (ISO/IEC 8859-1) 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 an ISO Latin-1 character. Table C.2 shows the escape sequences and their result.

Table C.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

C.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 will be used for the unspecified languages. Table C.3 shows examples of language codes.

Table C.3 – Using language codes in string literals

Language Code	Language
no code assigned	Default language. Is used if a language is requested that is not defined
|cn|	Chinese
|de|	German
|en|	English
|es|	Spanish
|fr|	French
|it|	Italian
|jp|	Japanese
|kr|	Korean

NOTE The default text without a language code should exist and is recommended in the English language.

Example The following string literal specifies the English phrase "Invalid Selection" in English and German:

"Invalid Selection"
"|de|Unzulässige Auswahl"

C.3 Operators

Table C.4 contains operators of the EDDL.

Table C.4 – EDDL Operators

!	!=	%	%=
&	&&	&=	(
)	*	*=	+
++	+=	,	-
--	-=	.	/
/=	:	;	<
<<	<<=	<=	=
==	>	>=	>>
>>=	?	[	]
^	^=	{	|
|=	||	}	~
->			

C.4 Keywords

Table C.5 contains the EDDL keywords.

Table C.5 – EDDL Keywords

EDDL keyword	EDDL keyword	EDDL keyword	EDDL keyword
!	COMM	HART	PROGRAMS
%	COMMAND	HEADER	PURPOSE
&	COMMANDS	HELP	READ
(	COMPUTATION	HIDDEN	READ_ONLY
)	CONNECTION	IF	READ_TIMEOUT
*	CONNECTIONS	if	RECORD
+	CONSTANT_UNIT	IGNORE_IN_TEMPORARY_MASTER	RECORDS
,	CONTAINED	IMPORT	REDEFINE
-	continue	INDEX	REDEFINE DOMAIN
.	CORRECTABLE	INFO	REDEFINITIONS
/	CORRECTION	INITIAL_VALUE	REFRESH
:	DATA	INPUT	REFRESH_ITEMS
;	DATA_ENTRY_ERROR	int	REFRESHS
<	DATA_ENTRY_WARNING	INTEGER	RELATIONS
=	DATA_EXCHANGE	ITEM_ARRAY	REPLY
>	DATE	ITEM_ARRAY_ITEMS	REQUEST
?	DATE_AND_TIME	ITEM_ARRAYS	RESPONSE_CODES
[	DD_REVISION	ITEMS	return
]	DEFAULT	LABEL	REVIEW
^	default	LAST	ROLE
{	DEFAULT_VALUE	LIKE	ROOT
|	DEFINITION	LOAD_TO_APPLICATION	SCALING_FACTOR
}	DELETE	LOAD_TO_DEVICE	SELECT
~	DELETE DOMAIN	LOCAL	SELF
!=	DETAIL	LOCAL_DISPLAY	SELF_CORRECTING
%=	DEVICE	long	SERVICE
&&	DEVICE_REVISION	MANUFACTURER	short
&=	DEVICE_TYPE	MANUFACTURER_EXT	signed

EDDL keyword	EDDL keyword	EDDL keyword	EDDL keyword
*=	DIAGNOSE	MAX_VALUE	SLOT
++	DIAGNOSTIC	MEMBERS	SOFTWARE
+=	DIALOG	MENU	STATE
--	DIGITAL_INPUT	MENU_ITEMS	STYLE
-=	DIGITAL_OUTPUT	MENUS	SUCCESS
->	DISCRETE_INPUT	METHOD	SUMMARY
/=	DISCRETE_OUTPUT	METHOD_ITEMS	switch
<<	DISPLAY_FORMAT	METHODS	TABLE
<=	DISPLAY_ITEMS	MIN_VALUE	TIME
==	DISPLAY_VALUE	MISC	TIME_VALUE
>=	do	MISC_ERROR	TRANSACTION
>>	DOMAIN	MISC_WARNING	TRANSDUCER
\0	DOMAINS	MODE	TRUE
^=	DOUBLE	MODE_ERROR	TUNE
|=	double	MODULE	TYPE
||	DURATION	MORE	UNCORRECTABLE
<<=	DYNAMIC	NEXT	UNIT
>>=	EDD_PROFILE	NUMBER	UNIT_ITEMS
ACCESS	EDD_REVISION	NUMBER_OF_ELEMENTS	UNITS
ADD	EDD_VERSION	OBJECT_REFERENCE	unsigned
ALARM	EDIT_DISPLAY	OCTET	UNSIGNED_INTEGER
ANALOG_INPUT	EDIT_DISPLAY_ITEMS	OF	VALIDITY
ANALOG_OUTPUT	EDIT_DISPLAYS	OFFLINE	VARIABLE
APPINSTANCE	EDIT_FORMAT	ONLINE	VARIABLE_LIST
ARGUMENTS	EDIT_ITEMS	OPERATE	VARIABLE_LISTS
ARRAY	ELEMENTS	OPERATION	VARIABLES
ARRAYS	ELSE	OUTPUT	VARIABLE_STATUS
ASCII	else	PACKED_ASCII	VARIABLE_STATUS_NONE
BAD	ENTRY	PARAMETER_LISTS	VARIABLE_STATUS_INVALID
BIT_ENUMERATED	ENUMERATED	PARAMETERS	VARIABLE_STATUS_NOT_ACCEPTED
BITSTRING	EUC	PARENT	VARIABLE_STATUS_NOT_SUPPORTED
BLOCK	EVENT	PASSWORD	VARIABLE_STATUS_CHANGED
BLOCKS	EVERYTHING	PHYSICAL	VARIABLE_STATUS_LOADED
break	FALSE	POST_EDIT_ACTIONS	VARIABLE_STATUS_INITIAL
CASE	FIRST	POST_READ_ACTIONS	VARIABLE_STATUS_NOT_CONVERTED
case	FLOAT	POST_WRITE_ACTIONS	VISIBLE
char	float	PRE_EDIT_ACTIONS	while
CHARACTERISTICS	for	PRE_READ_ACTIONS	WINDOW
CHILD	FREQUENCY_INPUT	PREV	WRITE
CLASS	FREQUENCY_OUTPUT	PRE_WRITE_ACTIONS	WRITE_AS_ONE
COLLECTION	FUNCTION	PROCESS	WRITE_AS_ONE_ITEMS
COLLECTION_ITEMS	HANDLING	PROCESS_ERROR	WRITE_AS_ONES
COLLECTIONS	HARDWARE	PROGRAM	WRITE_TIMEOUT

C.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                  = { '"?afnrtv'\' } .
       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 *  \"

/* character */
\' ISOLatin1char \'

/* Identifier */
```

C.6 letter (letter|digit|_)*Formal EDDL syntax

C.6.1 General

This annex contains a formal syntax of the EDDL using the Backus Naur Form.

If there are optional elements each element is marked with /* M */ if the element is mandatory or /* O */ if the element is optional.

C.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_listR

definition_listR
        = definition
        = definition_listR definition

definition
        = item
        = imported_description
        = like

item
        = array
        = block
        = collection
        = command
        = connection
        = domain
        = edit_display
        = item_array
        = menu
        = method
        = program
        = record
        = refresh_relation
        = response_codes_definition
        = unit_relation
        = variable
        = variable_list
        = wao_relation
```

C.6.3 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 */
        = block_a_collection_items                                  /* O */
        = block_a_edit_display_items                                /* O */
        = block_a_help                                              /* O */
        = block_a_item_array_items                                  /* O */
        = block_a_label                                             /* M */
        = block_a_menu_items                                        /* O */
        = block_a_method_items                                      /* O */
        = block_a_parameters                                        /* M */
        = block_a_parameter_lists                                   /* O */
        = block_a_refresh_items                                     /* O */
        = block_a_unit_items                                        /* O */
        = block_a_wao_items                                         /* O */

        = block_b_number                                            /* M */
        = block_b_type                                              /* M */

block_a_help
        = help

block_a_label
        = label

block_a_characteristics
        = 'CHARACTERISTICS' record_reference ';'

block_a_parameters
        = 'PARAMETERS' '{' member_list '}'

member_list
        = member
        = member_list member

block_a_parameter_lists
        = 'PARAMETER_LISTS' '{' 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_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_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_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_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_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_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_b_number
        = 'NUMBER' number_specifier

block_b_type
        = 'TYPE' 'PHYSICAL' ';'
        = 'TYPE' 'TRANSDUCER' ';'
        = 'TYPE' 'FUNCTION' ';'
```

C.6.4 COLLECTION

```
collection
        = 'COLLECTION' 'OF' collection_item_type Identifier '{'
          collection_attribute_list '}'

collection_item_type
        = item_type

item_type
        = 'ARRAY'
        = 'BLOCK'
        = 'COLLECTION' 'OF' item_type
        = 'COMMAND'
        = 'CONNECTION'
        = 'DOMAIN'
        = 'EDIT_DISPLAY'
        = 'ITEM_ARRAY' 'OF' item_type
        = 'MENU'
        = 'METHOD'
        = 'PROGRAM'
        = 'RECORD'
        = 'REFRESH'
        = 'RESPONSE_CODES'
        = 'UNIT'
        = 'VARIABLE'
        = 'VARIABLE_LIST'
        = 'WRITE_AS_ONE'

collection_attribute_list
        = collection_attribute
        = collection_attribute_list collection_attribute

collection_attribute
        = help                                                  /* O */
        = label                                                 /* O */
        = members                                               /* M */
```

C.6.5 COMMAND

```
command
        = 'COMMAND' Identifier '{' command_attribute_list '}'
        = 'COMMAND' Identifier '{' '}'

command_attribute_list
        = command_attribute
        = command_attribute_list command_attribute
```

```
command_attribute
        = command_header                                        /* O */
        = command_slot                                          /* O */
        = command_index                                         /* O */
        = command_block                                         /* O */
        = command_number                                        /* O */
        = command_operation                                     /* M */
        = command_transaction                                   /* M */
        = command_connection                                    /* O */
        = command_module                                        /* O */
        = command_response_codes                                /* O */

command_header
        = 'HEADER' header_spezifier

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_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_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_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_selection_list
        = number_selection
        = number_selection_list number_selection

number_selection
        = 'CASE' expr ':' number_specifier
        = 'DEFAULT' ':' number_specifier

command_operation
        = 'OPERATION' operation_specifier

operation_specifier
        = operation ';'
        = 'IF' '(' expr ')' '{' operation_specifier '}'
        = 'IF' '(' expr ')' '{' operation_specifier '}'
          'ELSE' '{' operation_specifier '}'
        = 'SELECT' '(' expr ')' '{' operation_selection_list '}'

operation_selection_list
        = operation_selection
        = operation_selection_list operation_selection

operation_selection
        = 'CASE' expr ':' operation_specifier
        = 'DEFAULT' ':' operation_specifier
```

```
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
        = data_items
        = data_items_list ',' data_items

data_items
        = integer
        = variable_reference
        = variable_reference '<' Integer '>'
        = variable_reference '(' data_items_qualifiers ')'
        = variable_reference '<' Integer '>' '(' data_items_qualifiers ')'

data_items_qualifiers
        = data_items_qualifiers_wrap

data_items_qualifiers_wrap
        = data_items_qualifier
        = data_items_qualifiers_wrap ',' data_items_qualifier

data_items_qualifier
        = 'INDEX'
        = 'INFO'

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_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

command_response_codes
        = response_codes
```

C.6.6 CONNECTION

```
connection
        = 'CONNECTION' Identifier '{' connection_attribute_list '}'

connection_attribute_list
        = connection_attribute
        = connection_attribute_list connection_attribute

connection_attribute
        = appinstance                                          /* M */

appinstance
        = 'APPINSTANCE' Integer ';'
```

C.6.7 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 */
```

C.6.8 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
        = display_items                                                 /* O */
        = edit_items                                                    /* M */
        = label                                                         /* M */
        = 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
```

C.6.9 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'
        = 'BLOCKS'
        = 'COLLECTIONS'
        = 'COMMANDS'
        = 'CONNECTIONS'
        = 'DOMAINS'
        = 'EDIT_DISPLAYS'
        = 'ITEM_ARRAYS'
        = 'MENUS'
        = 'METHODS'
        = 'PROGRAMS'
        = 'RECORDS'
        = 'REFRESHS'
        = 'RELATIONS'
        = 'RESPONSE_CODES'
        = 'UNITS'
        = 'VARIABLES'
        = 'VARIABLE_LISTS'
        = 'WRITE_AS_ONES'

redefinitions
        = 'REDEFINITIONS' '{' redefinition_list '}'
```

```
redefinition_list
        = redefinition
        = redefinition_list redefinition

redefinition
        = array_redefinition
        = block_redefinition
        = collection_redefinition
        = command_redefinition
        = connection_redefinition
        = domain_redefinition
        = edit_display_redefinition
        = item_array_redefinition
        = menu_redefinition
        = method_redefinition
        = program_redefinition
        = record_redefinition
        = refresh_relation_redefinition
        = response_codes_definition_redefinition
        = unit_relation_redefinition
        = variable_redefinition
        = variable_list_redefinition
        = wao_relation_redefinition
```

C.6.10 LIKE

```
like
        = Identifier 'LIKE' Identifier
          '{' attribute_redefinition_list '}'
        = Identifier 'LIKE' 'ARRAY' Identifier
          '{' array_attribute_redefinition_list '}'
        = Identifier 'LIKE' 'BLOCK' Identifier
          '{' block_attribute_redefinition_list '}'
        = Identifier 'LIKE' 'COLLECTION' 'OF' item_type          Identifier
          '{' collection_attribute_redefinition_list '}'
        = Identifier 'LIKE' 'COMMAND' Identifier
          '{' command_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' 'ITEM_ARRAY' 'OF' item_type
          Identifier
          '{' item_array_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
          '{' '}'
        = Identifier 'LIKE' 'RESPONSE_CODES' Identifier
          '{' response_code_redefinition_list '}'
        = Identifier 'LIKE' 'UNIT' Identifier
          '{' '}'
        = Identifier 'LIKE' 'VARIABLE' Identifier
          '{' variable_attribute_redefinition_list '}'
        = Identifier 'LIKE' 'VARIABLE_LIST' Identifier
          '{' variable_list_attribute_redefinition_list '}'
```

```
        = Identifier 'LIKE' 'WRITE_AS_ONE' Identifier
          '{' '}'

attribute_redefinition_list
        = attribute_redefinition
        = attribute_redefinition_list attribute_redefinition

attribute_redefinition
        = appinstance_redefinition
        = arguments_redefinition
        = array_type_redefinition
        = block_a_characteristics_redefinition
        = block_a_collection_item_redefinition
        = block_a_edit_disp_item_redefinition
        = block_a_item_array_item_redefinition
        = block_a_menu_item_redefinition
        = block_a_method_item_redefinition
        = block_a_parameters_redefinition
        = block_a_parameter_lists_redefinition
        = block_a_refresh_item_redefinition
        = block_a_unit_item_redefinition
        = block_a_wao_item_redefinition
        = block_b_number_redefinition
        = block_b_type_redefinition
        = command_slot_redefinition
        = command_index_redefinition
        = command_block_redefinition
        = command_operation_redefinition
        = command_transaction_redefinition
        = command_connection_redefinition
        = command_module_redefinition
        = display_items_redefinition
        = edit_items_redefinition
        = elements_redefinition
        = handling_redefinition
        = help_redefinition
        = members_redefinition
        = menu_access_redefinition
        = menu_entry_redefinition
        = menu_items_redefinition
        = menu_purpose_redefinition
        = menu_style_redefinition
        = method_definition_redefinition
        = number_of_elements_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
        = response_code_redefinition
        = response_codes_reference_redefinition
        = type_redefinition
        = validity_redefinition
        = variable_class_redefinition
        = write_timeout_redefinition
```

C.6.11 MENU

```
menu
        = 'MENU' Identifier '{' menu_attribute_list '}'

menu_attribute_list
        = menu_attribute
        = menu_attribute_list menu_attribute

menu_attribute
        = help                                                  /* O */
        = label                                                 /* M */
        = menu_access                                           /* O */
        = menu_entry                                            /* O */
        = menu_items                                            /* M */
        = menu_purpose                                          /* O */
        = menu_role                                             /* O */
        = menu_style                                            /* O */
        = validity                                              /* O */
        = pre_edit_actions                                      /* O */
        = pre_read_actions                                      /* O */
        = pre_write_actions                                     /* O */
        = post_edit_actions                                     /* O */
        = post_read_actions                                     /* O */
        = post_write_actions                                    /* O */

menu_access
        = 'ACCESS' 'ONLINE' ';'
        = 'ACCESS' 'OFFLINE' ';'

menu_entry
        = 'ENTRY' menu_entry_specifier

menu_entry_specifier
        = menu_entry_item ';'
        = 'IF' '(' expr ')' '{' menu_entry_specifier '}'
        = 'IF' '(' expr ')' '{' menu_entry_specifier '}'
          'ELSE' '{' menu_entry_specifier '}'
        = 'SELECT' '(' expr ')' '{' menu_entry_selection_list '}'

menu_entry_selection_list
        = menu_entry_selection
        = menu_entry_selection_list menu_entry_selection

menu_entry_selection
        = 'CASE' expr ':' menu_entry_specifier
        = 'DEFAULT' ':' menu_entry_specifier

menu_entry_item
        = 'ROOT'

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_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

menu_item_args
        = '(' argument_list ')'
        = '(' 'REVIEW' ')'
        = '(' variable_qualifier_list ')'

variable_qualifier_list
        = variable_qualifier
        = variable_qualifier_list ',' variable_qualifier

variable_qualifier
        = 'DISPLAY_VALUE'
        = 'READ_ONLY'
        = 'HIDDEN'

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_specifier

menu_style_specifier
        = menu_style_item ';'
        = 'IF' '(' expr ')' '{' menu_style_specifier '}'
        = 'IF' '(' expr ')' '{' menu_style_specifier '}'
          'ELSE' '{' menu_style_specifier '}'
        = 'SELECT' '(' expr ')' '{' menu_style_selection_list '}'

menu_style_selection_list
        = menu_style_selection
        = menu_style_selection_list menu_style_selection

menu_style_selection
        = 'CASE' expr ':' menu_style_specifier
        = 'DEFAULT' ':' menu_style_specifier

menu_style_item
        = 'WINDOW'
        = 'DIALOG'
        = string
```

C.6.12 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
        = 'int'
        = 'long'
        = 'float'

method_attribute_list
        = method_attribute
        = method_attribute_list method_attribute

method_attribute
        = help                                                  /* O */
        = label                                                 /* M */
        = method_access                                         /* O */
        = method_definition                                     /* M */
        = validity                                              /* O */
        = variable_class                                        /* M */

method_access
        = 'ACCESS' 'OFFLINE' ';'
        = 'ACCESS' 'ONLINE' ';'

method_definition
        = 'DEFINITION' c_compound_statement
```

C.6.13 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
```

C.6.14 RECORDS

```
record
        = 'RECORD' Identifier '{' record_attribute_list '}'

record_attribute_list
        = record_attribute
        = record_attribute_list record_attribute

record_attribute
        = help                                                  /* O */
        = label                                                 /* M */
        = members                                               /* M */
        = response_codes                                        /* O */
```

C.6.15 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_array_attribute_type   Identifier   '{'
item_array_attribute_list '}'
        =    'ARRAY'    'OF'    item_array_attribute_type    Identifier    '{'
item_array_attribute_list '}'

item_array_attribute_type
        = item_array_type

item_array_type
        = 'ARRAY'
        = 'BLOCK'
        = 'COLLECTION' 'OF' item_array_type
        = 'COMMAND'
        = 'CONNECTION'
        = 'DOMAIN'
```

```
        = 'ITEM_ARRAY' 'OF' item_array_type
        = 'MENU'
        = 'METHOD'
        = 'PROGRAM'
        = 'RECORD'
        = 'REFRESH'
        = 'RESPONSE_CODES'
        = 'UNIT'
        = 'VARIABLE'
        = 'VARIABLE_LIST'
        = 'WRITE_AS_ONE'

item_array_attribute_list
        = item_array_attribute
        = item_array_attribute_list item_array_attribute

item_array_attribute
        = elements                                              /* M */
        = help                                                  /* O */
        = label                                                 /* 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 ';'
```

C.6.16 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_reference_specifier_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
```

C.6.17 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'
```

C.6.18 VALUE_ARRAY

```
array
        = 'ARRAY' Identifier '{' array_attribute_list '}'

array_attribute_list
        = array_attribute
        = array_attribute_list array_attribute

array_attribute
        = array_type                                            /* M */
        = array_size                                            /* M */
        = help                                                  /* O */
        = label                                                 /* M */
        = response_codes                                        /* O */

array_type
        = 'TYPE' Identifier ';'

array_size
        = 'NUMBER_OF_ELEMENTS' Integer ';'
```

C.6.19 VARIABLE

```
variable
        = 'VARIABLE' Identifier '{' variable_attribute_list '}'

variable_attribute_list
        = variable_attribute
        = variable_attribute_list variable_attribute

variable_attribute
        = constant_unit                                         /* O */
        = handling                                              /* O */
        = help                                                  /* O */
        = label                                                 /* M */
        = 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 */
        = response_codes                                        /* O */
        = type                                                  /* M */
```

```
        = validity                                                  /* O */
        = variable_class                                            /* M */
        = variable_style                                            /* 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'
        = 'OUTPUT'
        = 'SERVICE'
        = 'TUNE'

variable_style
        = 'STYLE' string ';'

constant_unit
        = 'CONSTANT_UNIT' string_specifier

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_definition_
```

```
handling_definition_
        = handling_keyword
        = handling_definition_ '&' handling_keyword

handling_keyword
        = 'READ'
        = 'WRITE'

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 '}'

actions_specifier_list
        = actions_specifier
        = actions_specifier_list actions_specifier

actions_specifier
        = method_list
        = 'IF' '(' expr ')' '{' actions_specifier_list '}'
        = 'IF' '(' expr ')' '{' actions_specifier_list '}'
          'ELSE' '{' actions_specifier_list '}'
        = 'SELECT' '(' expr ')' '{' actions_selection_list '}'

actions_selection_list
        = actions_selection
        = actions_selection_list actions_selection

actions_selection
        = 'CASE' expr ':' actions_specifier_list
        = 'DEFAULT' ':' actions_specifier_list

method_list
        = method_specification
        = method_list ',' method_specification

method_specification
        = method_reference
        = method_definition

read_timeout
        = 'READ_TIMEOUT' expr_specifier

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

type
        = 'TYPE' type_specifier

type_specifier
        = arithmetic_type
        = enumerated_type
        = index_type
        = string_type
        = date_time_type
        = object_reference

arithmetic_type
        = 'INTEGER' arithmetic_option_size arithmetic_options
        = 'UNSIGNED_INTEGER' arithmetic_option_size arithmetic_options
        = 'DOUBLE' arithmetic_options
        = 'FLOAT' arithmetic_options

arithmetic_option_size
        =
        = '(' Integer ')'

arithmetic_options
        = ';'
        = '{' arithmetic_option_list '}'

arithmetic_option_list
        = arithmetic_option
        = arithmetic_option_list arithmetic_option

arithmetic_option
        = arithmetic_default_value
        = arithmetic_initial_value
        = display_format
        = edit_format
        = enumerator_list
        = maximum_value
        = maximum_value_n
        = minimum_value
        = minimum_value_n
        = scaling_factor

arithmetic_default_value
        = 'DEFAULT_VALUE' expr_specifier

arithmetic_initial_value
        = 'INITIAL_VALUE' expr_specifier
```

```
display_format
        = 'DISPLAY_FORMAT' string_specifier

edit_format
        = 'EDIT_FORMAT' string_specifier

scaling_factor
        = 'SCALING_FACTOR' expr_specifier

maximum_value
        = 'MAX_VALUE' expr_specifier

maximum_value_n
        = MAX_VALUE_n expr_specifier

minimum_value
        = 'MIN_VALUE' expr_specifier

minimum_value_n
        = MIN_VALUE_n expr_specifier

enumerated_type
        = 'ENUMERATED' enumerated_option_size '{' enumerated_option_list
          '}'
        = 'BIT_ENUMERATED' enumerated_option_size '{'
          bit_enumerated_option_list '}'

enumerated_option_size
        =
        = '(' Integer ')'

enumerated_option_list
        = enumerated_option
        = enumerated_option_list enumerated_option

enumerated_option
        = arithmetic_default_value
        = arithmetic_initial_value
        = enumerators_specifier

enumerators_specifier_list
        = enumerators_specifier
        = enumerators_specifier_list enumerators_specifier

enumerators_specifier
        = 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

enumerator_list
        = enumerator
        = enumerator_list ',' enumerator
```

```
enumerator
        = '{' enumerator_value ',' description_string '}'
        = '{' enumerator_value ',' description_string ',' help_string '}'

enumerator_value
        = Integer
        = Real
        = string

description_string
        = string

help_string
        = string

bit_enumerated_option_list
        = bit_enumerated_option
        = bit_enumerated_option_list bit_enumerated_option

bit_enumerated_option
        = arithmetic_default_value
        = arithmetic_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 ',' description_string
          '}'
        = '{' Integer ',' description_string ','
              help_string
          '}'
        = '{' Integer ',' description_string ','
              variable_class_definition_wrap
          '}'
        = '{' Integer ',' description_string ','
              status_class
          '}'
        = '{' Integer ',' description_string ','
              help_string ',' variable_class_definition_wrap
          '}'
```

```
        = '{' Integer ',' description_string ','
              help_string ',' status_class
          '}'
        = '{' Integer ',' description_string ','
              help_string ',' method_reference
          '}'
        = '{' Integer ',' description_string ','
              variable_class_definition_wrap ',' status_class
          '}'
        = '{' Integer ',' description_string ','
              variable_class_definition_wrap ',' method_reference
          '}'
        = '{' Integer ',' description_string ','
              status_class ',' method_reference
          '}'
        = '{' Integer ',' description_string ','
              help_string ',' variable_class_definition_wrap ','
status_class
          '}'
        = '{' Integer ',' description_string ','
              help_string     ','     variable_class_definition_wrap     ','
method_reference
          '}'
        = '{' Integer ',' description_string ','
              help_string ',' status_class ',' method_reference
          '}'
        = '{' Integer ',' description_string ','
              variable_class_definition_wrap     ','     status_class     ','
method_reference
          '}'
        = '{' Integer ',' description_string ','
              help_string     ','     variable_class_definition_wrap     ','
status_class ',' method_reference
          '}'

variable_class_definition_wrap
        = variable_class_definition

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'
```

```
index_type
        = 'INDEX' item_array_reference ';'
        = 'INDEX' '(' Integer ')' item_array_reference ';'

string_type
        = string_option_type string_option_size ';'
        = string_option_type string_option_size '{' '}'
        = string_option_type string_option_size '{' string_option_list '}'

string_option_type
        = string_keyword

string_keyword
        = 'ASCII'
        = 'BITSTRING'
        = 'EUC'
        = 'PACKED_ASCII'
        = 'PASSWORD'
        = 'VISIBLE'
        = 'OCTET'

string_option_size
        = '(' Integer ')'

string_option_list
        = string_option
        = string_option_list string_option

string_option
        = string_default_value
        = string_initial_value
        = enumerator_list

string_default_value
        = 'DEFAULT_VALUE' string ';'

string_initial_value
        = 'INITIAL_VALUE' string ';'

date_time_type
        = 'DATE' ';'
        = 'DATE' '{' '}'
        = 'DATE' '{' string_option_list '}'
        = 'DATE_AND_TIME' ';'
        = 'DATE_AND_TIME' '{' '}'
        = 'DATE_AND_TIME' '{' string_option_list '}'
        = 'DURATION' ';'
        = 'DURATION' '{' '}'
        = 'DURATION' '{' string_option_list '}'
        = 'TIME' ';'
        = 'TIME' '{' '}'
        = 'TIME' '{' string_option_list '}'
        = 'TIME' '(' Integer ')' ';'
        = 'TIME' '(' Integer ')' '{' '}'
        = 'TIME' '(' Integer ')' '{' string_option_list '}'
        = 'TIME_VALUE' ';'
        = 'TIME_VALUE' '{' '}'
        = 'TIME_VALUE' '{' string_option_list '}'

object_reference
        = 'OBJECT_REFERENCE' object_reference_value ';'
```

```
object_reference_value
        = 'ROOT'
        = 'PARENT'
        = 'CHILD'
        = 'SELF'
        = 'NEXT'
        = 'PREV'
        = 'FIRST'
        = 'LAST'
```

C.6.20 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
        = help                                                  /* O */
        = label                                                 /* O */
        = members                                               /* M */
        = response_codes                                        /* O */
```

C.6.21 Common attributes

```
help
        = 'HELP' string_specifier

label
        = 'LABEL' string_specifier

members
        = 'MEMBERS' '{' members_specifier_list '}'

number_specifier:
        = number_items ';'
        = 'IF' '(' expr ')' '{' number_specifier '}'
        = 'IF' '(' expr ')' '{' number_specifier '}'
          'ELSE' '{' number_specifier '}'
        = 'SELECT' '(' expr ')' '{' number_selection_list '}'

number_items:
        = expr
        = variable_reference

response_codes
        = 'RESPONSE_CODES' '{' response_codes_specifier_list '}'
        = 'RESPONSE_CODES' response_code_reference_specifier

validity
        = 'VALIDITY' boolean_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

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 ',' description_string ';'
        = Identifier ',' reference_without_call ',' description_string ','
help_string
          ';'

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

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'
```

C.6.22 OPEN, CLOSE

```
open
        = 'OPEN' filename

close
        = 'CLOSE' filename

filename
        = Identifier
```

C.6.23 Expression

```
primary_expr
        = reference
        = conditional_expr
        = String
        = '(' expr ')'

postfix_expr
        = primary_expr
        = postfix_expr '++'
        = postfix_expr '--'

unary_expr
        = postfix_expr
        = '++' unary_expr
        = '--' unary_expr
        = unary_operator unary_expr
        = Identifier '(' ')'
        = Identifier '(' c_argument_expr_list ')'
        = 'CURRENT_ROLE'
```

```
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

assignment_expr
        = conditional_expr
        = unary_expr assignment_operator assignment_expr
```

```
assignment_operator
        = '='
        = '*='
        = '/='
        = '%='
        = '+='
        = '-='
        = '>>='
        = '<<='
        = '&='
        = '^='
        = '|='

expr
        = assignment_expr
        = expr ',' assignment_expr
```

C.6.24 C-Grammer

```
c_primary_expr
        = c_char
        = c_dictionary
        = c_identifier
        = c_integer
        = c_real
        = c_string
        = '(' c_expr ')'

c_char
        = Character

c_dictionary
        = '[' Identifier ']'

c_identifier
        = Identifier

c_integer
        = ExprInteger

c_real
        = ExprReal

c_string
        = c_string_literal

c_string_literal
        = string_literal

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'

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 ';'
```

C.6.25 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_type_redefinition
        = array_size_redefinition
        = help_redefinition
        = required_label_redefinition
        = response_codes_reference_redefinition

array_type_redefinition
        = 'REDEFINE' array_type

array_size_redefinition
        = 'REDEFINE' array_size

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
        = block_a_collection_items_redefinition
        = block_a_edit_display_items_redefinition
        = block_a_help_redefinition
        = block_a_item_array_items_redefinition
        = block_a_label_redefinition
        = block_a_menu_items_redefinition
        = block_a_method_items_redefinition
        = block_a_parameters_redefinition
        = block_a_parameter_lists_redefinition
        = block_a_refresh_items_redefinition
        = block_a_unit_items_redefinition
        = block_a_wao_items_redefinition
        = block_b_number_redefinition
        = block_b_type_redefinition

block_a_characteristics_redefinition
        = 'REDEFINE' block_a_characteristics

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_help_redefinition
        = help_redefinition

block_a_item_array_items_redefinition
        = 'DELETE' 'ITEM_ARRAY_ITEMS' ';'
        = 'REDEFINE' block_a_item_array_items
```

```
block_a_label_redefinition
        = required_label_redefinition

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_parameters_redefinition
        = 'PARAMETERS' '{' parameter_redefinition_list '}'

parameter_redefinition_list
        = parameter_redefinition
        = parameter_redefinition_list parameter_redefinition

parameter_redefinition
        = 'REDEFINE' member
        = 'ADD' member

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_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_b_type_redefinition
        = 'REDEFINE' block_b_type

block_b_number_redefinition
        = 'REDEFINE' block_b_number

collection_redefinition
        = 'DELETE' 'COLLECTION' Identifier ';'
        = 'REDEFINE' 'COLLECTION' 'OF' item_type Identifier '{' '}'
        =   'REDEFINE'   'COLLECTION'   'OF'   item_type   Identifier   '{'
collection_attribute_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
        = help_redefinition
        = optional_label_redefinition
        = members_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_header_redefinition
        = command_slot_redefinition
        = command_index_redefinition
        = command_block_redefinition
        = command_number_redefinition
        = command_operation_redefinition
        = command_transaction_redefinition
        = command_connection_redefinition
        = command_module_redefinition
        = command_response_codes_redefinition

command_header_redefinition
        = 'DELETE' 'HEADER' ';'
        = 'REDEFINE' command_header

command_slot_redefinition
        = 'DELETE' 'SLOT' ';'
        = 'REDEFINE' command_slot

command_index_redefinition
        = 'DELETE' 'INDEX' ';'
        = 'REDEFINE' command_index

command_block_redefinition
        = 'DELETE' 'BLOCK' ';'
        = 'REDEFINE' command_block

command_number_redefinition
        = 'DELETE' 'NUMBER' ';'
        = 'REDEFINE' command_number

command_operation_redefinition
        = 'REDEFINE' command_operation

command_transaction_redefinition
        = 'REDEFINE' command_transaction

command_connection_redefinition
        = 'DELETE' 'CONNECTION' ';'
        = 'REDEFINE' command_connection

command_module_redefinition
        = 'DELETE' 'MODULE' ';'
        = 'REDEFINE' command_module

command_response_codes_redefinition
        = response_codes_reference_redefinition
```

```
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
        = display_items_redefinition
        = edit_items_redefinition
        = required_label_redefinition
        = post_edit_actions_redefinition
        = pre_edit_actions_redefinition

display_items_redefinition
        = 'DELETE' 'DISPLAY_ITEMS' ';'
        = 'REDEFINE' display_items

edit_items_redefinition
        = 'REDEFINE' edit_items
```

```
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

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

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
        = help_redefinition
        = required_label_redefinition
        = menu_access_redefinition
        = menu_entry_redefinition
        = menu_items_redefinition
        = menu_purpose_redefinition
        = menu_role_redefinition
        = menu_style_redefinition
        = validity_redefinition
        = pre_edit_actions_redefinition
        = pre_read_actions_redefinition
        = pre_write_actions_redefinition
        = post_edit_actions_redefinition
        = post_read_actions_redefinition
        = post_write_actions_redefinition

menu_access_redefinition
        = 'DELETE' 'ACCESS' ';'
        = 'REDEFINE' menu_access
```

```
menu_entry_redefinition
        = 'DELETE' 'ENTRY' ';'
        = 'REDEFINE' menu_entry

menu_items_redefinition
        = 'REDEFINE' menu_items

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
        = help_redefinition
        = required_label_redefinition
        = method_access_redefinition
        = method_definition_redefinition
        = validity_redefinition
        = variable_class_redefinition

method_access_redefinition
        = 'REDEFINE' method_access

method_definition_redefinition
        = 'REDEFINE' method_definition

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
        = help_redefinition
        = required_label_redefinition
        = members_redefinition
        = response_codes_reference_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

help_redefinition
        = 'DELETE' 'HELP' ';'
        = 'REDEFINE' help

required_label_redefinition
        = 'REDEFINE' label

optional_label_redefinition
        = 'DELETE' 'LABEL' ';'
        = 'REDEFINE' label
```

```
response_codes_reference_redefinition
        = 'DELETE' 'RESPONSE_CODES' ';'
        = 'REDEFINE' response_codes

validity_redefinition
        = 'DELETE' 'VALIDITY' ';'
        = 'REDEFINE' validity

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

variable_redefinition
        = 'DELETE' 'VARIABLE' Identifier ';'
        = 'REDEFINE' 'VARIABLE' Identifier '{' variable_attribute_list
          '}'
        = 'VARIABLE' Identifier '{' variable_attribute_redefinition_list
          '}'

variable_attribute_redefinition_list
        =
        = variable_attribute_redefinition_list
variable_attribute_redefinition

variable_attribute_redefinition
        = constant_unit_redefinition
        = handling_redefinition
        = help_redefinition
        = required_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
        = response_codes_reference_redefinition
        = type_redefinition
        = validity_redefinition
        = variable_class_redefinition
        = variable_style_redefinition
        = write_timeout_redefinition

variable_class_redefinition
        = 'REDEFINE' variable_class

variable_style_redefinition
        = 'DELETE' 'STYLE' ';'
        = 'REDEFINE' variable_style

constant_unit_redefinition
        = 'DELETE' 'CONSTANT_UNIT' ';'
        = 'REDEFINE' constant_unit
```

```
handling_redefinition
        = 'DELETE' 'HANDLING' ';'
        = 'REDEFINE' handling

pre_edit_actions_redefinition
        = 'DELETE' 'PRE_EDIT_ACTIONS' ';'
        = 'REDEFINE' pre_edit_actions

post_edit_actions_redefinition
        = 'DELETE' 'POST_EDIT_ACTIONS' ';'
        = 'REDEFINE' post_edit_actions

pre_read_actions_redefinition
        = 'DELETE' 'PRE_READ_ACTIONS' ';'
        = 'REDEFINE' pre_read_actions

post_read_actions_redefinition
        = 'DELETE' 'POST_READ_ACTIONS' ';'
        = 'REDEFINE' post_read_actions

pre_write_actions_redefinition
        = 'DELETE' 'PRE_WRITE_ACTIONS' ';'
        = 'REDEFINE' pre_write_actions

post_write_actions_redefinition
        = 'DELETE' 'POST_WRITE_ACTIONS' ';'
        = 'REDEFINE' post_write_actions

read_timeout_redefinition
        = 'DELETE' 'READ_TIMEOUT' ';'
        = 'REDEFINE' read_timeout

write_timeout_redefinition
        = 'DELETE' 'WRITE_TIMEOUT' ';'
        = 'REDEFINE' write_timeout

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


arithmetic_type_redefinition
        = 'INTEGER' arithmetic_option_size_redefinition '{'
          arithmetic_option_redefinition_list '}'
        = 'UNSIGNED_INTEGER' arithmetic_option_size_redefinition '{'
          arithmetic_option_redefinition_list '}'
        = 'FLOAT' '{' arithmetic_option_redefinition_list '}'
        = 'DOUBLE' '{' arithmetic_option_redefinition_list '}'

arithmetic_option_size_redefinition
        = arithmetic_option_size_redefinition1

arithmetic_option_size_redefinition1
        =
        = '(' Integer ')'
```

```
arithmetic_option_redefinition_list
        = arithmetic_option_redefinition
        =                                   arithmetic_option_redefinition_list
arithmetic_option_redefinition

arithmetic_option_redefinition
        = arithmetic_default_value_redefinition
        = arithmetic_initial_value_redefinition
        = display_format_redefinition
        = edit_format_redefinition
        = enumerator_list_redefinition
        = maximum_value_redefinition
        = maximum_value_n_redefinition
        = minimum_value_redefinition
        = minimum_value_n_redefinition
        = scaling_factor_redefinition

arithmetic_default_value_redefinition
        = 'DELETE' 'DEFAULT_VALUE' ';'
        = 'REDEFINE' arithmetic_default_value

arithmetic_initial_value_redefinition
        = 'DELETE' 'INITIAL_VALUE' ';'
        = 'REDEFINE' arithmetic_initial_value

display_format_redefinition
        = 'DELETE' 'DISPLAY_FORMAT' ';'
        = 'REDEFINE' display_format

edit_format_redefinition
        = 'DELETE' 'EDIT_FORMAT' ';'
        = 'REDEFINE' edit_format

enumerator_list_redefinition
        = 'DELETE' Integer ';'
        = 'REDEFINE' enumerator
        = 'ADD' enumerator

maximum_value_redefinition
        = 'DELETE' 'MAX_VALUE' ';'
        = 'REDEFINE' maximum_value

maximum_value_n_redefinition
        = 'DELETE' MAX_VALUE_n ';'
        = 'REDEFINE' maximum_value_n

minimum_value_redefinition
        = 'DELETE' 'MIN_VALUE' ';'
        = 'REDEFINE' minimum_value

minimum_value_n_redefinition
        = 'DELETE' MIN_VALUE_n ';'
        = 'REDEFINE' minimum_value_n

scaling_factor_redefinition
        = 'DELETE' 'SCALING_FACTOR' ';'
        = 'REDEFINE' scaling_factor

enumerated_type_redefinition
        = 'ENUMERATED' enumerated_option_size_redefinition '{'
          enumerated_option_redefinition_list '}'
        = 'BIT_ENUMERATED' enumerated_option_size_redefinition '{'
          bit_enumerated_option_redefinition_list '}'
```

```
enumerated_option_size_redefinition
        = enumerated_option_size_redefinition1

enumerated_option_size_redefinition1
        =
        = '(' Integer ')'

enumerated_option_redefinition_list
        = enumerated_option_redefinition
        =                                enumerated_option_redefinition_list
enumerated_option_redefinition

enumerated_option_redefinition
        = arithmetic_default_value_redefinition
        = arithmetic_initial_value_redefinition
        = enumerators_redefinition

enumerators_redefinition
        = 'DELETE' Integer ';'
        = 'REDEFINE' enumerator
        = 'ADD' enumerator

bit_enumerated_option_redefinition_list
        = bit_enumerated_option_redefinition
        =                              bit_enumerated_option_redefinition_list
bit_enumerated_option_redefinition

bit_enumerated_option_redefinition
        = arithmetic_default_value_redefinition
        = arithmetic_initial_value_redefinition
        = bit_enumerators_redefinition

bit_enumerators_redefinition
        = 'DELETE' Integer ';'
        = 'REDEFINE' bit_enumerator
        = 'ADD' bit_enumerator

string_type_redefinition
        = string_option_type '{' string_option_redefinition_list '}'
        =    string_option_type     string_option_size_redefinition     '{'
string_option_redefinition_list '}'

string_option_size_redefinition
        = string_option_size_redefinition1

string_option_size_redefinition1
        = '(' Integer ')'

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_list_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_list_redefinition
        = 'DELETE' string ';'
        = 'REDEFINE' enumerator
        = 'ADD' enumerator

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
        = help_redefinition
        = optional_label_redefinition
        = members_redefinition
        = response_codes_reference_redefinition
```

C.6.26 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

selector
        = 'CONSTANT_UNIT'
        = 'DEFAULT_VALUE'
        = 'HELP'
        = 'INITIAL_VALUE'
        = 'LABEL'
        = 'MAX_VALUE'
        = MAX_VALUE_n
        = 'MIN_VALUE'
        = MIN_VALUE_n
        = 'SCALING_FACTOR'
        = 'VARIABLE_STATUS'

dictionary_reference
        = '[' Identifier ']'

block_reference
        = reference_without_call

collection_reference
        = reference_without_call
```

```
connection_reference
        = reference_without_call

edit_display_reference
        = reference_without_call

item_array_reference
        = reference_without_call

menu_reference
        = reference_without_call

method_reference
        = reference_without_call

record_reference
        = reference_without_call

refresh_reference
        = reference_without_call

response_code_reference
        = reference_without_call

unit_reference
        = reference_without_call

variable_reference
        = reference_without_call

wao_reference
        = reference_without_call
```

Annex D
(normative)

EDDL Builtin Library

D.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.

D.2 Conventions for Builtin descriptions

The following conventions apply for the Builtin definitions:

- Table D.1 defines the format for the Builtins lexical element tables
- Table D.2 defines the contents of the columns of Table D.1.

Table D.1 – Format for the Builtins lexical element tables

Parameter Name	Type	Direction	Usage	Description

Table D.2 – Contents of the lexical element table

Column	Text	Meaning
Parameter Name	<name>	is the name of the formal parameter
	"<return>"	represents the return value of the Builtin
Type	data type	abstract data type (basic or derived) of the formal parameter
Direction	I	specifies an input parameter
	O	specifies an output parameter
	I/O	specifies an input and output parameter
	—	to be used if there is no returned value (VOID)
Usage	m	formal parameter is mandatory
	o	formal parameter is optional
	c	formal parameter is conditional
	—	to be used if there is no returned value (VOID)
Description	<text>	descriptive text for the parameter
	—	to be used if there is no returned value (VOID)

D.3 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 D.3.

Table D.3 – Builtin abort

Parameter Name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

D.4 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 D.4.

Table D.4 – Builtin abort_on_all_comm_errors

Parameter Name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

D.5 Builtin ABORT_ON_ALL_COMM_STATUS

Purpose

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 D.5.

Table D.5 – Builtin ABORT_ON_ALL_COMM_STATUS

Parameter Name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

D.6 Builtin ABORT_ON_ALL_DEVICE_STATUS

Purpose

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 D.6.

Table D.6 – Builtin ABORT_ON_ALL_DEVICE_STATUS

Parameter Name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

D.7 Builtin ABORT_ON_ALL_RESPONSE_CODES

Purpose

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 D.7.

Table D.7 – Builtin ABORT_ON_ALL_RESPONSE_CODES

Parameter Name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

D.8 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 D.8.

Table D.8 – Builtin abort_on_all_response_codes

Parameter Name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

D.9 Builtin abort_on_comm_error

Purpose

Causes the method to abort upon receiving a specific communications error.

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 D.9.

Table D.9 – Builtin abort_on_comm_error

Parameter Name	Type	Direction	Usage	Description
error	unsigned long	I	m	specifies the error code
<return>	long	O	m	is any of the return codes specified in Table D.198

D.10 Builtin ABORT_ON_COMM_ERROR

Purpose

Builtin ABORT_ON_COMM_ERROR will set the no comm error mask such 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 D.10.

Table D.10 – Builtin ABORT_ON_COMM_ERROR

Parameter Name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

D.11 Builtin ABORT_ON_COMM_STATUS

Purpose

The Builtin ABORT_ON_COMM_STATUS will set the correct bit(s) in the comm status abort mask such 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 l.

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 D.11.

Table D.11 – Builtin ABORT_ON_COMM_STATUS

Parameter Name	Type	Direction	Usage	Description
comm_status	unsigned char	I	m	specifies a bit mask for the communication status
<return>	VOID	—	—	—

D.12 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 D.12.

Table D.12 – Builtin ABORT_ON_DEVICE_STATUS

Parameter Name	Type	Direction	Usage	Description
device_status	unsigned char	I	m	specifies a bit mask for the device status
<return>	VOID	—	—	—

D.13 Builtin ABORT_ON_NO_DEVICE

Purpose

The Builtin ABORT_ON_NO_DEVICE will set the no devices mask such 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 D.13.

Table D.13 – Builtin ABORT_ON_NO_DEVICE

Parameter Name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

D.14 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

```
void ABORT_ON_RESPONSE_CODE(response code)
```

The lexical elements of the Builtin ABORT_ON_RESPONSE_CODE are shown in Table D.14.

Table D.14 – Builtin ABORT_ON_RESPONSE_CODE

Parameter Name	Type	Direction	Usage	Description
response_code	int	I	m	specifies a bit mask for the response code
<return>	VOID	—	—	—

D.15 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.

10.1 Lexical structure

```
abort_on_response_code(code)
```

The lexical elements of the Builtin abort_on_response_code are shown in Table D.15.

Table D.15 – Builtin abort_on_response_code

Parameter Name	Type	Direction	Usage	Description
code	unsigned long	I	m	specifies a bit mask for the response code
<return>	long	O	m	is any of the return codes specified in Table D.198

D.16 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 D.16.

Table D.16 – Builtin ACKNOWLEDGE

Parameter Name	Type	Direction	Usage	Description
prompt	char[]	I	m	specifies a message to be displayed
<return>	int	O	m	specifies the return value of the Builtin

NOTE The symbolic name for the return value should be SUCCESS.

D.17 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 D.17.

Table D.17 – Builtin acknowledge

Parameter Name	Type	Direction	Usage	Description
prompt	char[]	I	m	specifies a message to be displayed
global_var_ids	array of int	I	m	specifies an array which contains unique identifiers of VARIABLE instance
<return>	int	O	m	specifies the return value of the Builtin

NOTE The symbolic name for the return value should be SUCCESS.

D.18 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 D.18.

Table D.18 – Builtin add_abort_method

Parameter Name	Type	Direction	Usage	Description
abort_method_id	reference	I	m	specifies the name of a METHOD instance
<return>	int	O	m	specifies whether the method was successfully added to the list or if the list was full

NOTE The symbolic names for the return value should be TRUE or FALSE.

D.19 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's execution 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. Builtins abort methods cannot call add_abort_method.

Lexical structure

```
long
add_abort_method (method_id)
```

The lexical elements of the Builtin add_abort_method are shown in Table D.19.

Table D.19 – Builtin add_abort_method

Parameter Name	Type	Direction	Usage	Description
method_id	unsigned long	I	m	specifies the ID of the method
<return>	long	O	m	is any of the return codes specified in Table D.198

D.20 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

```
long assign (dst_id, dst_member_id, src_id, src_member_id)
```

The lexical elements of the Builtin assign are shown in Table D.20.

Table D.20 – Builtin assign

Parameter Name	Type	Direction	Usage	Description
dst_id	unsigned long	I	m	specifies the item ID of the destination variable
dst_member_id	unsigned long	I	m	specifies the member ID of the destination variable
src_id	unsigned long	I	m	specifies the item ID of the source variable
src_member_id	unsigned long	I	m	specifies the member ID of the source variable
<return>	long	O	m	is any of the return codes specified in Table D.198

D.21 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 D.21.

Table D.21 – Builtin assign_double

Parameter Name	Type	Direction	Usage	Description
device_var	reference	I	m	specifies the name of the variable
new_value	double	I	m	specifies the new value of the referenced variable
<return>	int	O	m	specifies the return value of the Builtin

NOTE The symbolic name for the return value should be TRUE if the assignment was successful and FALSE if the variable identifier was invalid.

D.22 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 D.22.

Table D.22 – Builtin assign_float

Parameter Name	Type	Direction	Usage	Description
device_var	reference	I	m	specifies the name of the variable
new_value	double	I	m	specifies the new value of the referenced variable
<return>	int	O	m	specifies the return value of the Builtin

NOTE The symbolic name for the return value should be TRUE if the assignment was successful and FALSE if the variable identifier was invalid.

D.23 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 D.23.

Table D.23 – Builtin assign_int

Parameter Name	Type	Direction	Usage	Description
device_var	reference	I	m	specifies the name of the variable
new_value	int	I	m	specifies the new value of the referenced variable
<return>	int	O	m	specifies the return value of the Builtin

NOTE The symbolic name for the return value should be TRUE if the assignment was successful and FALSE if the variable identifier was invalid.

D.24 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 D.24.

Table D.24 – Builtin assign_var

Parameter Name	Type	Direction	Usage	Description
destination_var	reference	I	m	specifies the name of the destination variable
source_var	reference	I	m	specifies the name of the source variable
<return>	int	O	m	specifies the return value of the Builtin

NOTE The symbolic name for the return value should be TRUE if the assignment was successful and FALSE if the variable identifier was invalid

D.25 Builtin atof

Purpose

This Builtin converts a text into a floating-point value.

Lexical structure

`atof(str)`

The lexical elements of the Builtin atof are shown in Table D.25.

Table D.25 – Builtin atof

Parameter Name	Type	Direction	Usage	Description
str	char[]	I	m	specifies the text to convert
<return>	float	O	m	specifies the return value of the Builtin

D.26 Builtin atoi

Purpose

This Builtin converts a text into an integer value.

Lexical structure

`atoi(str)`

The lexical elements of the Builtin atoi are shown in Table D.26.

Table D.26 – Builtin atoi

Parameter Name	Type	Direction	Usage	Description
str	char[]	I	m	specifies the text to convert
<return>	int	O	m	specifies the return value of the Builtin

D.27 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 D.27.

Table D.27 – Builtin dassign

Parameter Name	Type	Direction	Usage	Description
device_var	reference	I	m	specifies the name of the variable
new_value	double	I	m	specifies the new value of the referenced variable
<return>	int	O	m	specifies the return value of the Builtin

NOTE The symbolic name for the return value should be TRUE if the assignment was successful and FALSE if the variable identifier was invalid.

D.28 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 D.29

Table D.28 – Builtin Date_to_DayOfMonth

Parameter Name	Type	Direction	Usage	Description
date	long	I	m	specifies 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	returns the day of month

D.29 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 D.29.

Table D.29 – Builtin Date_to_Month

Parameter Name	Type	Direction	Usage	Description
date	long	I	m	specifies 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	returns the month

D.30 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 D.30.

Table D.30 – Builtin Date_to_Year

Parameter Name	Type	Direction	Usage	Description
date	long	I	m	specifies 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	returns the month from 1 January 1900

D.31 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 D.31.

Table D.31 – Builtin DELAY

Parameter Name	Type	Direction	Usage	Description
delay_time	unsigned integer	I	m	specifies the delay in number of seconds
prompt	char[]	I	m	specifies a message to be displayed
<return>	VOID	—	—	—

D.32 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 D.32.

Table D.32 – Builtin delay

Parameter Name	Type	Direction	Usage	Description
delay_time	unsigned integer	I	m	specifies the delay in number of seconds
prompt	char[]	I	m	specifies a message to be displayed
global_var_ids	array of int	I	m	specifies an array which contains unique identifiers of VARIABLE instance
<return>	VOID	—	—	—

D.33 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 D.33.

Table D.33 – Builtin DELAY_TIME

Parameter Name	Type	Direction	Usage	Description
delay_time	unsigned integer	I	m	specifies the delay in number of seconds
<return>	VOID	—	—	—

D.34 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 D.34.

Table D.34 – Builtin delayfor

Parameter Name	Type	Direction	Usage	Description
duration	long	I	m	specifies the delay in number of seconds
prompt	char[]	I	m	specifies a message to be displayed
ids	array of unsigned long	I	m	specifies the item IDs of a VARIABLE instance used in the prompt message
indices	array of unsigned long	I	m	specifies 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	specifies 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	is any of the return codes specified in Table D.198

D.35 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 D.35.

Table D.35 – Builtin DICT_ID

Parameter Name	Type	Direction	Usage	Description
name	char[]	I	m	specifies the name of the dictionary
<return>	long	O	m	specifies dictionary ID for the named string

D.36 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 D.36.

Table D.36 – Builtin discard_on_exit

Parameter Name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

D.37 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 D.37.

Table D.37 – Builtin display

Parameter Name	Type	Direction	Usage	Description
prompt	char[]	I	m	specifies a message to be displayed
global_var_ids	array of int	I	m	specifies an array which contains unique identifiers of VARIABLE instance
<return>	VOID	—	—	—

D.38 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 D.38.

Table D.38 – Builtin display_builtin_error

Parameter Name	Type	Direction	Usage	Description
error	long	I	M	specifies a code for a Builtin error
<return>	long	O	m	is any of the return codes specified in Table D.198

D.39 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 D.39.

Table D.39 – Builtin display_comm_error

Parameter Name	Type	Direction	Usage	Description
error	unsigned long	I	M	specifies a code for a communication error
<return>	long	O	m	is any of the return codes specified in Table D.198

D.40 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 D.40.

Table D.40 – Builtin display_comm_status

Parameter Name	Type	Direction	Usage	Description
comm_status_value	int	I	M	specifies a code for a communication error
<return>	VOID	—	—	—

D.41 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 D.41.

Table D.41 – Builtin display_device_status

Parameter Name	Type	Direction	Usage	Description
device_status_value	int	I	M	specifies a code for the device status
<return>	VOID	—	—	—

D.42 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 D.42.

Table D.42 – Builtin display_dynamics

Parameter Name	Type	Direction	Usage	Description
prompt	char[]	I	m	specifies a message to be displayed
ids	array of unsigned long	I	m	specifies the item IDs of the VARIABLEs used in the prompt message
indices	array of unsigned long	I	m	specifies 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	specifies 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	is any of the return codes specified in Table D.198

D.43 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 D.43.

Table D.43 – Builtin display_message

Parameter Name	Type	Direction	Usage	Description
prompt	char[]	I	m	specifies a message to be displayed
ids	array of unsigned long	I	m	specifies the item IDs of the VARIABLEs used in the prompt message
indices	array of unsigned long	I	m	specifies 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	specifies 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	is any of the return codes specified in Table D.198

D.44 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 D.44.

Table D.44 – Builtin display_response_code

Parameter Name	Type	Direction	Usage	Description
id	unsigned long	I	m	specifies the item ID of the item which generated the response code
member_id	unsigned long	I	m	specifies the member ID of the item which generated the response code
code	unsigned long	I	m	specifies the response code as received from Builtin get_response_code
<return>	long	O	m	is any of the return codes specified in Table D.198

D.45 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 D.45.

Table D.45 – Builtin display_response_status

Parameter Name	Type	Direction	Usage	Description
response_code_value	int	I	m	specifies the response code to display in a prompt
<return>	VOID	—	—	—

D.46 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 D.46.

Table D.46 – Builtin display_xmtr_status

Parameter Name	Type	Direction	Usage	Description
xmtr_variable	int	I	m	specifies a reference to a VARIABLE instance
response_code_value	int	I	m	specifies a response code value
<return>	VOID	—	—	—

D.47 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 D.47

Table D.47 – Builtin edit_device_value

Parameter Name	Type	Direction	Usage	Description
prompt	char[]	I	m	specifies a message to be displayed
ids	array of unsigned long	I	m	specifies the item IDs of the VARIABLEs used in the prompt message
indices	array of unsigned long	I	m	specifies 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	specifies 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	specifies the item ID of the VARIABLE to be modified
param_member_id	unsigned long	I	m	specifies the member ID of the VARIBLE to be modified
<return>	long	O	m	is any of the return codes specified in Table D.198

D.48 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 D.48.

Table D.48 – Builtin edit_local_value

Parameter Name	Type	Direction	Usage	Description
prompt	char[]	I	m	specifies a message to be displayed
ids	array of unsigned long	I	m	specifies the item IDs of the VARIABLEs used in the prompt message
indices	array of unsigned long	I	m	specifies 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	specifies 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	specifies the name of the local VARIABLE that may be modified
<return>	long	O	m	is any of the return codes specified in Table D.198

D.49 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 D.49.

Table D.49 – Builtin ext_send_command

Parameter Name	Type	Direction	Usage	Description
cmd_number	int	I	m	specifies the COMMAND number
cmd_status	char[]	I	m	specifies 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	specifies the return value of the Builtin

NOTE The symbolic name for the return value should be BI_SUCESS if the command was successful, BI_COMM_ERROR if an error occurred sending the command, or BI_NO_DEVICE if no device was found.

D.50 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 D.50.

Table D.50 – Builtin ext_send_command_trans

Parameter Name	Type	Direction	Usage	Description
cmd_number	int	I	m	specifies the COMMAND number
transaction	int	I	m	specifies the number of the TRANSACTION
cmd_status	char[]	I	m	specifies 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	specifies the return value of the Builtin

NOTE The symbolic name for the return value should be BI_SUCESS if the command was successful, BI_COMM_ERROR if an error occurred sending the command, or BI_NO_DEVICE if no device was found.

D.51 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 D.51.

Table D.51 – Builtin fail_on_all_comm_errors

Parameter Name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

D.52 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 D.52.

Table D.52 – Builtin fail_on_all_response_codes

Parameter Name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

D.53 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 D.53.

Table D.53 – Builtin fail_on_comm_error

Parameter Name	Type	Direction	Usage	Description
error	unsigned long	I	m	specifies a communication error code
<return>	long	O	m	is any of the return codes specified in Table D.198

D.54 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 D.54.

Table D.54 – Builtin fail_on_response_code

Parameter Name	Type	Direction	Usage	Description
code	unsigned long	I	m	specifies the response code
<return>	long	O	m	is any of the return codes specified in Table D.198

D.55 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 D.55.

Table D.55 – Builtin fassign

Parameter Name	Type	Direction	Usage	Description
device_var	reference	I	m	specifies the name of the VARIABLE
new_value	double	I	m	specifies the new value of the referenced VARIABLE
<return>	int	O	m	specifies the return value of the Builtin

NOTE The symbolic name for the return value should be TRUE if the assignment was successful, and FALSE if the variable identifier was invalid.

D.56 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 D.56.

Table D.56 – Builtin fgetval

Parameter Name	Type	Direction	Usage	Description
<return>	double	O	m	specifies the value of the variable

D.57 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 D.57.

Table D.57 – Builtin float_value

Parameter Name	Type	Direction	Usage	Description
source_var_name	reference	I	m	specifies the name of a VARIABLE reference
<return>	double	O	m	specifies the value of the variable

D.58 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 D.58.

Table D.58 – Builtin fsetval

Parameter Name	Type	Direction	Usage	Description
value	double	I	m	specifies the new value for the VARIABLE
<return>	double	O	m	specifies the value of the variable

D.59 Builtin ftoa

Purpose

This 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 D.59.

Table D.59 – Builtin ftoa

Parameter Name	Type	Direction	Usage	Description
f	float	I	m	specifies a floating point value to convert to a text
<return>	char[]	O	m	specifies the returned text

D.60 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 D.60.

Table D.60 – Builtin fvar_value

Parameter Name	Type	Direction	Usage	Description
source_var	reference	I	m	specifies the name of the VARIABLE reference
<return>	double	O	m	specifies the value of the variable

D.61 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 D.61.

Table D.61 – Builtin get_acknowledgement

Parameter Name	Type	Direction	Usage	Description
prompt	char[]	I	m	specifies a message to be displayed
ids	array of unsigned long	I	m	specifies the item IDs of the VARIABLEs used in the prompt message
indices	array of unsigned long	I	m	specifies 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
id_count	long	I	m	specifies 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	is any of the return codes specified in Table D.198

D.62 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 D.62.

Table D.62 – Builtin get_comm_error

Parameter Name	Type	Direction	Usage	Description
<return>	unsigned long	O	m	specifies a communication error code

D.63 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 D.63.

Table D.63 – Builtin get_comm_error_string

Parameter Name	Type	Direction	Usage	Description
error	unsigned long	I	m	specifies the communication error code
string	char[]	I	m	specifies a description of the communication error
maxlen	long	I	m	specifies the maximal length of string
<return>	long	O	m	is any of the return codes specified in Table D.198

D.64 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 D.64.

Table D.64 – Builtin get_date

Parameter Name	Type	Direction	Usage	Description
data	char[]	I	m	specifies the returned value of the VARIABLE. The structure of this octet buffer is defined in this specification. This output parameters shall point to valid storage
size	long	I	m	specifies the returned length of the VARIABLE. This output parameter shall point to valid storage
<return>	long	O	m	is any of the return codes specified in Table D.198

D.65 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 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_value(id, member_id, data, size)
```

The lexical elements of the Builtin get_date_value are shown in Table D.65.

Table D.65 – Builtin get_date_value

Parameter Name	Type	Direction	Usage	Description
id	unsigned long	I	m	specifies an ID of a VARIABLE reference
member_id	unsigned long	I	m	specifies a member ID of a BLOCK_A, PARAMETER_LIST, RECORD, VALUE_ARRAY, COLLECTION or REFERENCE_ARRAY
data	char[]	I	m	specifies a buffer of max. 8 octet for the result of the Builtin
size	long	I	m	specifies the size of the used buffer
<return>	long	O	m	is any of the return codes specified in Table D.198

D.66 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 D.66.

Table D.66 – Builtin get_dds_error

Parameter Name	Type	Direction	Usage	Description
string	char[]	I	m	specifies a pointer to a text description of the error
maxlen	long	I	m	specifies 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	specifies an error code generated by an earlier Builtin call

D.67 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 D.67.

Table D.67 – Builtin GET_DEV_VAR_VALUE

Parameter Name	Type	Direction	Usage	Description
prompt	char[]	I	m	specifies a message to be displayed
device_var_name	reference	I	m	specifies the referenced name of the VARIABLE, which can be edited
<return>	int	O	m	specifies the return value of the Builtin

NOTE The symbolic name for the return value should be BI_SUCCESS if the variable was successfully modified, or BI_ERROR if an error occurred entering the new value or accessing the specified variable.

D.68 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 D.68.

Table D.68 – Builtin get_dev_var_value

Parameter Name	Type	Direction	Usage	Description
prompt	char[]	I	m	specifies a message to be displayed
global_var_ids	array of int	I	m	specifies an array which contains unique identifiers of VARIABLE instance
device_var_name	reference	I	m	specifies a name of the VARIABLE, which can be edited
<return>	int	O	m	specifies a status if the VARIABLE was successfully modified or if an error occurred entering the new value or accessing the specified VARIABLE

NOTE The symbolic name for the return value should be BI_SUCCESS if the variable was successfully modified, or BI_ERROR if an error occurred entering the new value or accessing the specified variable.

D.69 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 D.69.

Table D.69 – Builtin get_dictionary_string

Parameter Name	Type	Direction	Usage	Description
dict_string_name	reference	I	m	specifies the name of a dictionary entry
string	char[]	I	m	specifies a buffer for the result string
max_str_len	int	I	m	specifies the maximum length of the buffer
<return>	int	O	m	specifies a status if successful

NOTE The symbolic name for the return value should be TRUE if successful, FALSE if string could not be found.

D.70 Builtin get_double

Purpose

The Builtin get_double retrieves the current un-scaled 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 D.70.

Table D.70 – Builtin get_double

Parameter Name	Type	Direction	Usage	Description
value	double	O	m	specifies the returned value of the VARIABLE
<return>	long	O	m	is any of the return codes specified in Table D.198

D.71 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 D.71.

Table D.71 – Builtin get_double_value

Parameter Name	Type	Direction	Usage	Description
id	unsigned long	I	m	specifies the Item ID of the VARIABLE to access
member_id	unsigned long	I	m	specifies the member ID of the VARIABLE to access
value	double	O	m	specifies the returned value of the VARIABLE
<return>	long	O	m	is any of the return codes specified in Table D.198

D.72 Builtin get_float

Purpose

The Builtin get_float retrieves the current un-scaled 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 D.72

Table D.72 – Builtin get_float

Parameter Name	Type	Direction	Usage	Description
value	float	O	m	specifies the returned value of the VARIABLE
<return>	long	O	m	is any of the return codes specified in Table D.198

D.73 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 D.73.

Table D.73 – Builtin get_float_value

Parameter Name	Type	Direction	Usage	Description
id	unsigned long	I	m	specifies the item ID of the VARIABLE to access
member_id	unsigned long	I	m	specifies the member ID of the VARIABLE to access
value	float	O	m	specifies the returned value of the VARIABLE
<return>	long	O	m	is any of the return codes specified in Table D.198

D.74 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 D.74.

Table D.74 – Builtin GET_LOCAL_VAR_VALUE

Parameter Name	Type	Direction	Usage	Description
prompt	char[]	I	m	specifies a message to be displayed
local_var_name	referenced	I	m	specifies the name of the local VARIABLE to edit
<return>	int	O	m	specifies a status if the VARIABLE was successfully modified or if an error occurred entering the new value or accessing the specified VARIABLE

NOTE The symbolic name for the return value should be BI_SUCCESS if the variable was successfully modified, or BI_ERROR if an error occurred entering the new value or accessing the specified variable.

D.75 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 D.75.

Table D.75 – Builtin get_local_var_value

Parameter Name	Type	Direction	Usage	Description
prompt	char[]	I	m	specifies a message to be displayed
global_var_ids	array of int	I	o	specifies an array which contains unique identifiers of VARIABLE instance
local_var_name	reference d	I	o	specifies the name of the local VARIABLE to edit
<return>	int	O	m	specifies a status if the VARIABLE was successfully modified or if an error occurred entering the new value or accessing the specified VARIABLE

NOTE The symbolic name for the return value should be BI_SUCCESS if the variable was successfully modified, or BI_ERROR if an error occurred entering the new value or accessing the specified variable.

D.76 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 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 D.76.

Table D.76 – 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	specifies a status if the command was successfully, if an error occurred or no device was found

NOTE The symbolic name for the return value should be BI_SUCESS if the command was successful, BI_COMM_ERROR if an error occurred sending the command, or BI_NO_DEVICE if no device was found.

D.77 Builtin get_resolve_status

Purpose

The Builtin get_resolve_status return 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 D.77.

Table D.77 – Builtin get_resolve_status

Parameter Name	Type	Direction	Usage	Description
<return>	unsigned long	O	m	specifies an error number

D.78 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, which 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 D.78.

Table D.78 – Builtin get_response_code

Parameter Name	Type	Direction	Usage	Description
resp_code	unsigned long	I	m	specifies the number of the returned response code
err_id	unsigned long	O	m	specifies the returned item ID of the item generating the response code
err_member_id	unsigned long	O	m	specifies the returned member ID of the item causing a response code
<return>	long	O	m	is any of the return codes specified in Table D.198

D.79 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 D.79.

Table D.79 – Builtin get_response_code_string

Parameter Name	Type	Direction	Usage	Description
id	unsigned long	I	m	specifies the item ID of the item to find
member_id	unsigned long	I	m	specifies the member ID of the item to find
code	unsigned long	I	m	specifies the response code for which the descriptive string is to be found
string	char*	O	m	specifies the buffer for the returned response code description
maxlen	long	I	m	specifies the maximum length of the buffer
<return>	long	O	m	is any of the return codes specified in Table D.198

D.80 Builtin get_signed

Purpose

The Builtin get_signed retrieves the current un-scaled 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 D.80.

Table D.80 – Builtin get_signed

Parameter Name	Type	Direction	Usage	Description
value	long	I	m	specifies the returned value of the signed VARIABLE
<return>	long	O	m	is any of the return codes specified in Table D.198

D.81 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 D.81.

Table D.81 – Builtin get_signed_value

Parameter Name	Type	Direction	Usage	Description
id	unsigned long	I	m	specifies the item ID of the VARIABLE to access
member_id	unsigned long	I	m	specifies the member ID of the VARIABLE to access.
value	long	O	m	specifies the returned value of the signed VARIABLE
<return>	long	O	m	is any of the return codes specified in Table D.198

D.82 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 D.82.

Table D.82 – Builtin get_status_code_string

Parameter Name	Type	Direction	Usage	Description
variable_name	char[]	I	m	specifies the name of the VARIABLE to access
status_code	int	I	m	specifies the value of interesting status code
status_string	char *	O	m	specifies the buffer of the returned value of the signed VARIABLE
status_string_length	int	I	m	specifies the maximum size of the buffer
<return>	VOID	—	—	—

D.83 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 BITE NUMERATED 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 D.83.

Table D.83 – Builtin get_status_string

Parameter Name	Type	Direction	Usage	Description
id	unsigned long	I	m	specifies the item ID of the VARIABLE to access
member_id	unsigned long	I	m	specifies the member ID of the VARIABLE to access
status	unsigned long	I	m	specifies the value to search
string	char*	O	m	specifies the buffer for the returned string
maxlen	long	I	m	specifies the maximum size of the buffer
<return>	long	O	m	is any of the return codes specified in Table D.198

D.84 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 D.84.

Table D.84 – Builtin get_stddict_string

Parameter Name	Type	Direction	Usage	Description
id	unsigned long	I	m	specifies the text dictionary identifier
string	char*	O	m	specifies the returned buffer for the string
maxlen	long	I	m	specifies the maximum size of the buffer
<return>	long	O	m	is any of the return codes specified in Table D.198

D.85 Builtin get_string

Purpose

The Builtin get_string retrieves the current, un-scaled, 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 D.85.

Table D.85 – Builtin get_string

Parameter Name	Type	Direction	Usage	Description
string	char*	O	m	specifies the buffer for the returned string
len	long	I	m	specifies the maximum size of the string. The returned length of the string
<return>	int	O	m	is any of the return codes specified in Table D.198

D.86 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 D.86.

Table D.86 – Builtin get_string_value

Parameter Name	Type	Direction	Usage	Description
id	unsigned long	I	m	specifies the ID of a VARIABLE reference
member_id	unsigned long	I	m	specifies 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	specifies the size of the result string
<return>	long	O	m	is any of the return codes specified in Table D.198

D.87 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 D.87.

Table D.87 – Builtin GET_TICK_COUNT

Parameter Name	Type	Direction	Usage	Description
<return>	long	O	m	specifies the time in ms since the last system boot

D.88 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 D.88.

Table D.88 – Builtin get_unsigned

Parameter Name	Type	Direction	Usage	Description
value	unsigned long	O	m	specifies the returned value
<return>	long	O	m	is any of the return codes specified in Table D.198

D.89 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 D.89.

Table D.89 – Builtin get_unsigned_value

Parameter Name	Type	Direction	Usage	Description
id	unsigned long	I	m	specifies the item ID of the variable to access
member_id	unsigned long	I	m	specifies the member ID of the variable to access
value	unsigned long	O	m	specifies the returned value of the variable
<return>	long	O	m	is any of the return codes specified in Table D.198

D.90 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 D.90.

Table D.90 – Builtin iassign

Parameter Name	Type	Direction	Usage	Description
device_var	reference	I	m	specifies the name of the variable to access
new_value	int	I	m	specifies the new value of the VARIABLE
<return>	int	O	m	specifies the return value of the Builtin

NOTE The symbolic name for the return value should be TRUE if the assignment was successful, and FALSE if the variable identifier was invalid.

D.91 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 D.91.

Table D.91 – Builtin igetval

Parameter Name	Type	Direction	Usage	Description
<return>	int	O	m	specifies the value of the variable

D.92 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 D.92.

Table D.92 – Builtin IGNORE_ALL_COMM_STATUS

Parameter Name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

D.93 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 D.93.

Table D.93 – Builtin IGNORE_ALL_DEVICE_STATUS

Parameter Name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

D.94 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 D.94.

Table D.94 – Builtin IGNORE_ALL_RESPONSE_CODES

Parameter Name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

D.95 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 D.95.

Table D.95 – Builtin IGNORE_COMM_ERROR

Parameter Name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

D.96 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 D.96.

Table D.96 – Builtin IGNORE_COMM_STATUS

Parameter Name	Type	Direction	Usage	Description
comm_status	int	I	m	specifies the new retry and abort mask for ignoring
<return>	VOID	—	—	—

D.97 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 D.97.

Table D.97 – Builtin IGNORE_DEVICE_STATUS

Parameter Name	Type	Direction	Usage	Description
device_status	int	I	m	specifies the new retry and abort mask for ignoring
<return>	VOID	—	—	—

D.98 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 D.98.

Table D.98 – Builtin IGNORE_NO_DEVICE

Parameter Name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

D.99 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

```
void IGNORE_RESPONSE_CODE(response_code)
```

The lexical elements of the Builtin IGNORE_RESPONSE_CODE are shown in Table D.99.

Table D.99 – 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	—	—	—

D.100 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 D.100.

Table D.100 – Builtin int_value

Parameter Name	Type	Direction	Usage	Description
source_var_name	char[]	I	m	specifies the name of the VARIABLE
<return>	int	O	m	specifies the value of the variable

D.101 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 D.101.

Table D.101 – Builtin is_NaN

Parameter Name	Type	Direction	Usage	Description
dvalue	double	I	m	specifies the name of the variable
<return>	long	O	m	is one if the dvalue is equal to NaN, zero otherwise

D.102 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 D.102.

Table D.102 – Builtin isetval

Parameter Name	Type	Direction	Usage	Description
value	int	I	m	specifies the new value of the variable
<return>	int	O	m	specifies the value of the variable

D.103 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 D.103.

Table D.103 – Builtin ITEM_ID

Parameter Name	Type	Direction	Usage	Description
name	reference	I	m	specifies the name of the variable
<return>	unsigned long	O	m	specifies the VARIABLEs item ID

D.104 Builtin itoa

Purpose

This Builtin converts a floating-point value into a text string.

Lexical structure

`itoa(i)`

The lexical elements of the Builtin itoa are shown in Table D.104.

Table D.104 – Builtin itoa

Parameter Name	Type	Direction	Usage	Description
i	int	I	m	specifies an integer value to convert to a text
<return>	char[]	O	m	specifies the returned text

D.105 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 D.105.

Table D.105 – Builtin ivar_value

Parameter Name	Type	Direction	Usage	Description
source_var_name	reference	I	m	specifies the name of the variable
<return>	int	O	m	specifies the value of the variable

D.106 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 D.106.

Table D.106 – Builtin lassign

Parameter Name	Type	Direction	Usage	Description
var_name	reference	I	m	specifies the name of the VARIABLE
new_value	int	I	m	specifies the new value of the variable
<return>	int	O	m	specifies the return value of the Builtin

NOTE The symbolic name for the return value should be TRUE if the assignment was successful, and FALSE if the variable identifier was invalid.

D.107 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 D.107.

Table D.107 – Builtin lgetval

Parameter Name	Type	Direction	Usage	Description
<return>	int	O	m	specifies the value of the variable

D.108 Builtin LOG_MESSAGE

Purpose

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 D.108.

Table D.108 – Lexical elements of Builtin LOG_MESSAGE

Parameter Name	Type	Direction	Usage	Description
priority	int	I	m	specifies the priority of the message. 0: error, 1: warning, 2: low priority message
message	char[]	I	m	specifies the message string to insert into the logging protocol
<return>	VOID	O	m	—

D.109 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 D.109.

Table D.109 – Builtin long_value

Parameter Name	Type	Direction	Usage	Description
source_var_name	reference	I	m	specifies the name of the variable
<return>	int	O	m	specifies the value of the variable

D.110 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 D.110.

Table D.110 – Builtin lsetval

Parameter Name	Type	Direction	Usage	Description
value	int	I	m	specifies the new value of the VARIABLE
<return>	int	O	m	specifies the value of the variable

D.111 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 D.111.

Table D.111 – Builtin lvar_value

Parameter Name	Type	Direction	Usage	Description
source_var_name	reference	I	m	specifies the name of the VARIABLE
<return>	int	O	m	specifies the value of the variable

D.112 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 D.112.

Table D.112 – Builtin MEMBER_ID

Parameter Name	Type	Direction	Usage	Description
name	reference	I	m	specifies the name of an item member
<return>	unsigned long	O	m	specifies the referenced member ID

D.113 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 D.113.

Table D.113 – Builtin method_abort

Parameter Name	Type	Direction	Usage	Description
prompt	char[]	I	m	specifies a message to be displayed
<return>	VOID	O	m	

D.114 Builtin ObjectReference

Purpose

The Builtin ObjectReference returns unique id of an object. With the return object reference it's 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 D.114.

Table D.114 – Builtin process_abort

Parameter Name	Type	Direction	Usage	Description
String	char[]	I	m	Direction of navigation or Name of an object, see Table 103
<return>	long	O	m	Specifies an ID which represent an object reference

D.115 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 D.115.

Table D.115 – Builtin process_abort

Parameter Name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

D.116 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 D.116.

Table D.116 – Builtin put_date

Parameter Name	Type	Direction	Usage	Description
data	char[]	I	m	specifies the new value of the VARIABLE
size	long	I	m	specifies the length of the new value
<return>	long	O	m	is any of the return codes specified in Table D.198

D.117 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 D.117.

Table D.117 – Builtin put_date_value

Parameter Name	Type	Direction	Usage	Description
id	unsigned long	I	o	specifies the item ID of the VARIABLE to access
member_id	unsigned long	I	o	specifies the member ID of the VARIABLE access
data	char[]	I	m	specifies the new value of the VARIABLE
size	long	I	m	specifies the length of the new value
<return>	long	O	m	is any of the return codes specified in Table D.198

D.118 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 D.118.

Table D.118 – Builtin put_double

Parameter Name	Type	Direction	Usage	Description
value	double	I	m	specifies the new value of the VARIABLE
<return>	long	O	m	is any of the return codes specified in Table D.198

D.119 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.

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_value(id, member_id, value)`

The lexical elements of the Builtin put_double_value are shown in Table D.119.

Table D.119 – Builtin put_double_value

Parameter Name	Type	Direction	Usage	Description
id	unsigned long	I	o	specifies the item ID of the VARIABLE to access
member_id	unsigned long	I	o	specifies the member ID of the VARIABLE access
value	double	I	m	specifies the new value of the VARIABLE
<return>	long	O	m	is any of the return codes specified in Table D.198

D.120 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 D.120.

Table D.120 – Builtin put_float

Parameter Name	Type	Direction	Usage	Description
value	double	I	m	specifies the new value of the VARIABLE
<return>	long	O	m	is any of the return codes specified in Table D.198

D.121 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 D.121.

Table D.121 – Builtin put_float_value

Parameter Name	Type	Direction	Usage	Description
id	unsigned long	I	o	specifies the item ID of the VARIABLE to access
member_id	unsigned long	I	o	specifies the member ID of the VARIABLE access
value	double	I	m	specifies the new value of the VARIABLE
<return>	long	O	m	is any of the return codes specified in Table D.198

D.122 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 D.122.

Table D.122 – Builtin PUT_MESSAGE

Parameter Name	Type	Direction	Usage	Description
message	char[]	I	m	specifies a message to be displayed
<return>	VOID	—	—	—

D.123 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 variable's current value.

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 D.123.

Table D.123 – Builtin put_message

Parameter Name	Type	Direction	Usage	Description
message	char[]	I	m	specifies a message to be displayed
global_var_ids	array of int	I	m	specifies an array which contains unique identifiers of VARIABLE instance
<return>	VOID	—	—	—

D.124 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 D.124.

Table D.124 – Builtin put_signed

Parameter Name	Type	Direction	Usage	Description
value	long	I	m	specifies the new value of the VARIABLE
<return>	long	O	m	is any of the return codes specified in Table D.198

D.125 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 D.125.

Table D.125 – Builtin put_signed_value

Parameter Name	Type	Direction	Usage	Description
id	unsigned long	I	o	specifies the item ID of the VARIABLE to access
member_id	unsigned long	I	o	specifies the member ID of the VARIABLE access
value	long	I	m	specifies the new value of the VARIABLE
<return>	long	O	m	is any of the return codes specified in Table D.198

D.126 Builtin put_string

Purpose

The Builtin put_string stores the new un-scaled 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 D.126.

Table D.126 – Builtin put_string

Parameter Name	Type	Direction	Usage	Description
string	char[]	I	m	specifies the new value of the VARIABLE
len	long	I	m	specifies the length of the new value
<return>	long	O	m	is any of the return codes specified in Table D.198

D.127 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 D.127.

Table D.127 – Builtin put_string_value

Parameter Name	Type	Direction	Usage	Description
id	unsigned long	I	o	specifies the item ID of the VARIABLE to access
member_id	unsigned long	I	o	specifies the member ID of the VARIABLE access
string	char[]	I	m	specifies the new value of the VARIABLE
len	long	I	m	specifies the length of the new value
<return>	long	O	m	is any of the return codes specified in Table D.198

D.128 Builtin put_unsigned

Purpose

The Builtin put_unsigned stores the new un-scaled 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 D.128.

Table D.128 – Builtin put_unsigned

Parameter Name	Type	Direction	Usage	Description
value	unsigned long	I	o	specifies the new value of the VARIABLE
<return>	long	O	m	is any of the return codes specified in Table D.198

D.129 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 D.129.

Table D.129 – Builtin put_unsigned_value

Parameter Name	Type	Direction	Usage	Description
id	unsigned long	I	o	specifies the item ID of the VARIABLE to access
member_id	unsigned long	I	o	specifies the member ID of the VARIABLE access
value	unsigned long	I	o	specifies the new value of the VARIABLE
<return>	long	O	m	is any of the return codes specified in Table D.198

D.130 Builtin READ_COMMAND

Purpose

Builtin READ_COMMAND allows reading variables with a given command.

Lexical structure

```
READ_COMMANR(command_identifier)
```

The lexical elements of the Builtin READ_COMMAND are shown in Table D.130.

Table D.130 – Lexical elements of Builtin READ_COMMAND

Parameter Name	Type	Direction	Usage	Description
command_identifier	reference	I	m	specifies the Identifier of the command to execute
<return>	VOID	—	—	—

D.131 Builtin read_value

Purpose

The Builtin read_value 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_value(id, member_id)
```

The lexical elements of the Builtin read_value are shown in Table D.131.

Table D.131 – Builtin read_value

Parameter Name	Type	Direction	Usage	Description
id	unsigned long	I	o	specifies the item ID of the VARIABLE to access
member_id	unsigned long	I	o	specifies the member ID of the VARIABLE access
<return>	long	O	m	is any of the return codes specified in Table D.198

D.132 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 D.132.

Table D.132 – Builtin remove_abort_method

Parameter Name	Type	Direction	Usage	Description
abort_method_name	reference	I	m	specifies the identifier of the method
<return>	int	O	m	specifies the return value of the Builtin

NOTE The symbolic name for the return value should be TRUE if the method was successfully removed from the list, and FALSE if either the method was not in the list or if this function was run during an abort method.

D.133 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 D.133.

Table D.133 – Builtin remove_abort_method

Parameter Name	Type	Direction	Usage	Description
method_id	unsigned long	I	m	specifies the ID of the method
<return>	long	O	m	is any of the return codes specified in Table D.198

D.134 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 D.134.

Table D.134 – Builtin remove_all_abort_methods

Parameter Name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

D.135 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 D.135.

Table D.135 – Builtin resolve_array_ref

Parameter Name	Type	Direction	Usage	Description
id	unsigned long	I	m	specifies the item ID of the REFERENCE_ARRAY
member_id	unsigned long	I	m	specifies the index of the requested REFERENCE_ARRAY element
<return>	unsigned long	O	m	specifies 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

D.136 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 D.136.

Table D.136 – Builtin resolve_block_ref

Parameter Name	Type	Direction	Usage	Description
member_id	unsigned long	I	m	specifies the member ID of BLOCK_A
<return>	unsigned long	O	m	specifies 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

D.137 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 D.137.

Table D.137 – Builtin resolve_param_list_ref

Parameter Name	Type	Direction	Usage	Description
member_id	unsigned long	I	m	specifies the member ID of the parameter list
<return>	unsigned long	O	m	specifies 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

D.138 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 D.138.

Table D.138 – Builtin resolve_parm_ref

Parameter Name	Type	Direction	Usage	Description
member_id	unsigned long	I	m	specifies the member ID of the parameter
<return>	unsigned long	O	m	specifies 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

D.139 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 D.139.

Table D.139 – Builtin resolve_record_ref

Parameter Name	Type	Direction	Usage	Description
id	unsigned long	I	m	specifies the item ID of the RECORD being resolved
member_id	unsigned long	I	m	specifies the member ID of the record member
<return>	unsigned long	O	m	specifies 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

D.140 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 D.140.

Table D.140 – Builtin retry_on_all_comm_errors

Parameter Name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

D.141 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 D.141.

Table D.141 – Builtin RETRY_ON_ALL_COMM_STATUS

Parameter Name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

D.142 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 D.142.

Table D.142 – Builtin RETRY_ON_ALL_DEVICE_STATUS

Parameter Name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

D.143 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 D.143.

Table D.143 – Builtin RETRY_ON_ALL_RESPONSE_CODES

Parameter Name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

D.144 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 D.144.

Table D.144 – Builtin retry_on_all_response_codes

Parameter Name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

D.145 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 D.145.

Table D.145 – Builtin RETRY_ON_COMM_ERROR

Parameter Name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

D.146 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 D.146.

Table D.146 – Builtin retry_on_comm_error

Parameter Name	Type	Direction	Usage	Description
error	unsigned long	I	m	specifies the communication error
<return>	long	O	m	is any of the return codes specified in Table D.198

D.147 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 D.147.

Table D.147 – Builtin RETRY_ON_COMM_STATUS

Parameter Name	Type	Direction	Usage	Description
comm_status	int	I	m	specifies the new comm status retry mask
<return>	VOID	—	—	—

D.148 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 D.148.

Table D.148 – Builtin RETRY_ON_DEVICE_STATUS

Parameter Name	Type	Direction	Usage	Description
device_status	int	I	m	specifies the new comm status retry mask
<return>	VOID	—	—	—

D.149 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 D.149.