

INTERNATIONAL STANDARD

NORME INTERNATIONALE

**Function blocks (FB) for process control and electronic device description
language (EDDL) –
Part 5: EDDL Built-in library**

**Blocs fonctionnels (FB) pour les procédés industriels et le langage de
description électronique de produit (EDDL) –
Partie 5: Bibliothèque de Built-in EDDL**



THIS PUBLICATION IS COPYRIGHT PROTECTED

Copyright © 2015 IEC, Geneva, Switzerland

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either IEC or IEC's member National Committee in the country of the requester. If you have any questions about IEC copyright or have an enquiry about obtaining additional rights to this publication, please contact the address below or your local IEC member National Committee for further information.

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

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

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

About the IEC

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

About IEC publications

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

IEC Catalogue - webstore.iec.ch/catalogue

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

IEC publications search - www.iec.ch/searchpub

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

IEC Just Published - webstore.iec.ch/justpublished

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

Electropedia - www.electropedia.org

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

IEC Glossary - std.iec.ch/glossary

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

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

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

A propos de l'IEC

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

A propos des publications IEC

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

Catalogue IEC - webstore.iec.ch/catalogue

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

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

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

IEC Just Published - webstore.iec.ch/justpublished

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

Electropedia - www.electropedia.org

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

Glossaire IEC - std.iec.ch/glossary

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

Service Clients - webstore.iec.ch/csc

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

INTERNATIONAL STANDARD

NORME INTERNATIONALE

**Function blocks (FB) for process control and electronic device description
language (EDDL) –
Part 5: EDDL Built-in library**

**Blocs fonctionnels (FB) pour les procédés industriels et le langage de
description électronique de produit (EDDL) –
Partie 5: Bibliothèque de Built-in EDDL**

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

COMMISSION
ELECTROTECHNIQUE
INTERNATIONALE

ICS 25.040.40; 35.240.50

ISBN 978-2-8322-2702-2

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

CONTENTS

FOREWORD.....	19
INTRODUCTION.....	21
1 Scope.....	22
2 Normative references.....	22
3 Terms, definitions, acronyms and abbreviated terms	22
3.1 Terms and definitions	22
3.2 Acronyms and abbreviated terms.....	22
4 EDDL Builtin library	23
4.1 General.....	23
4.2 Conventions for Builtin descriptions	23
4.3 Builtin categories	24
4.3.1 Overview	24
4.3.2 User interface Builtins	25
4.3.3 Communication Builtins	26
4.3.4 Action Builtins	28
4.4 Builtin _ERROR	28
4.5 Builtin _TRACE	29
4.6 Builtin _WARNING	29
4.7 Builtin abort	29
4.8 Builtin abort_on_all_comm_errors.....	30
4.9 Builtin ABORT_ON_ALL_COMM_STATUS.....	30
4.10 Builtin ABORT_ON_ALL_DEVICE_STATUS.....	31
4.11 Builtin ABORT_ON_ALL_RESPONSE_CODES	31
4.12 Builtin abort_on_all_response_codes.....	32
4.13 Builtin ABORT_ON_COMM_ERROR.....	32
4.14 Builtin abort_on_comm_error.....	33
4.15 Builtin ABORT_ON_COMM_STATUS.....	33
4.16 Builtin ABORT_ON_DEVICE_STATUS.....	34
4.17 Builtin ABORT_ON_NO_DEVICE.....	34
4.18 Builtin ABORT_ON_RESPONSE_CODE	35
4.19 Builtin abort_on_response_code.....	36
4.20 Builtin abortTransferPort	36
4.21 Builtin abs.....	37
4.22 Builtin ACKNOWLEDGE	37
4.23 Builtin acknowledge	37
4.24 Builtin acos	38
4.25 Builtin add_abort_method (version A)	38
4.26 Builtin add_abort_method (version B)	38
4.27 Builtin AddTime.....	39
4.28 Builtin asin	39
4.29 Builtin assign	40
4.30 Builtin assign_double	40
4.31 Builtin assign_float.....	41
4.32 Builtin assign_int.....	41
4.33 Builtin assign_var.....	41
4.34 Builtin assign2	42
4.35 Builtin atan.....	42

4.36	Builtin atof	43
4.37	Builtin atoi.....	43
4.38	Builtin browselidentity	43
4.39	Builtin BUILD_MESSAGE	44
4.40	Builtin ByteToDouble.....	44
4.41	Builtin ByteToFloat.....	45
4.42	Builtin ByteToLong	45
4.43	Builtin ByteToShort	45
4.44	Builtin cbrt	46
4.45	Builtin ceil	46
4.46	Builtin closeTransferPort	46
4.47	Builtin cos	47
4.48	Builtin cosh	47
4.49	Builtin dassign	47
4.50	Builtin DATE_AND_TIME_VALUE_to_string.....	48
4.51	Builtin Date_to_DayOfMonth.....	48
4.52	Builtin DATE_to_days.....	49
4.53	Builtin Date_to_Month	49
4.54	Builtin DATE_to_string	49
4.55	Builtin Date_To_Time	50
4.56	Builtin Date_to_Year	50
4.57	Builtin days_to_DATE.....	50
4.58	Builtin DELAY	51
4.59	Builtin delay	51
4.60	Builtin DELAY_TIME	52
4.61	Builtin delayfor	52
4.62	Builtin delayfor2	53
4.63	Builtin DICT_ID	53
4.64	Builtin dictionary_string	54
4.65	Builtin DiffTime	54
4.66	Builtin discard_on_exit	55
4.67	Builtin DISPLAY	55
4.68	Builtin display.....	56
4.69	Builtin display_bitenum.....	56
4.70	Builtin display_builtin_error	57
4.71	Builtin display_comm_error	57
4.72	Builtin display_comm_status.....	58
4.73	Builtin display_device_status.....	58
4.74	Builtin display_dynamics	58
4.75	Builtin display_dynamics2	59
4.76	Builtin display_message	60
4.77	Builtin display_message2	61
4.78	Builtin display_response_code	61
4.79	Builtin display_response_status.....	62
4.80	Builtin display_xmtr_status	62
4.81	Builtin DoubleToByte.....	63
4.82	Builtin drand.....	63
4.83	Builtin dseed	64
4.84	Builtin edit_device_value	64

4.85	Builtin edit_device_value2	65
4.86	Builtin edit_local_value	66
4.87	Builtin edit_local_value2	67
4.88	Builtin exp	68
4.89	Builtin ext_send_command	68
4.90	Builtin ext_send_command_trans	69
4.91	Builtin fail_on_all_comm_errors	70
4.92	Builtin fail_on_all_response_codes	70
4.93	Builtin fail_on_comm_error	71
4.94	Builtin fail_on_response_code	71
4.95	Builtin fassign	72
4.96	Builtin fGetByte	72
4.97	Builtin fgetval	72
4.98	Builtin float_value	73
4.99	Builtin FloatToByte	73
4.100	Builtin floor	74
4.101	Builtin fmod	74
4.102	Builtin fpclassify	74
4.103	Builtin From_DATE_AND_TIME_VALUE	75
4.104	Builtin From_TIME_VALUE	75
4.105	Builtin fsetval	76
4.106	Builtin ftoa	76
4.107	Builtin fvar_value	76
4.108	Builtin get_acknowledgement	77
4.109	Builtin get_acknowledgement2	77
4.110	Builtin get_block_instance_by_object_index	78
4.111	Builtin get_block_instance_by_tag	78
4.112	Builtin get_block_instance_count	79
4.113	Builtin get_comm_error	79
4.114	Builtin get_comm_error_string	80
4.115	Builtin get_date	80
4.116	Builtin get_date_lelem	81
4.117	Builtin get_date_lelem2	81
4.118	Builtin get_date_value	82
4.119	Builtin get_date_value2	82
4.120	Builtin GET_DD_REVISION	83
4.121	Builtin get_dds_error	83
4.122	Builtin GET_DEV_VAR_VALUE	84
4.123	Builtin get_dev_var_value	84
4.124	Builtin GET_DEVICE_REVISION	85
4.125	Builtin GET_DEVICE_TYPE	85
4.126	Builtin get_dictionary_string	86
4.127	Builtin get_double	86
4.128	Builtin get_double_lelem	86
4.129	Builtin get_double_lelem2	87
4.130	Builtin get_double_value	87
4.131	Builtin get_double_value2	88
4.132	Builtin get_enum_string	89
4.133	Builtin get_float	89

4.134	Builtin get_float_lelem	89
4.135	Builtin get_float_lelem2	90
4.136	Builtin get_float_value	90
4.137	Builtin get_float_value2	91
4.138	Builtin GET_LOCAL_VAR_VALUE	91
4.139	Builtin get_local_var_value	92
4.140	Builtin GET_MANUFACTURER	92
4.141	Builtin get_more_status	93
4.142	Builtin get_resolve_status	93
4.143	Builtin get_response_code	94
4.144	Builtin get_response_code_string	94
4.145	Builtin get_rspcode_string	95
4.146	Builtin get_rspcode_string_by_id	95
4.147	Builtin get_signed	96
4.148	Builtin get_signed_lelem	96
4.149	Builtin get_signed_lelem2	97
4.150	Builtin get_signed_value	97
4.151	Builtin get_signed_value2	98
4.152	Builtin get_status_code_string	98
4.153	Builtin get_status_string	99
4.154	Builtin get_stddict_string	99
4.155	Builtin get_string	100
4.156	Builtin get_string_lelem	101
4.157	Builtin get_string_lelem2	101
4.158	Builtin get_string_value	102
4.159	Builtin get_string_value2	102
4.160	Builtin GET_TICK_COUNT	103
4.161	Builtin get_transfer_status	103
4.162	Builtin get_unsigned	104
4.163	Builtin get_unsigned_lelem	104
4.164	Builtin get_unsigned_lelem2	105
4.165	Builtin get_unsigned_value	105
4.166	Builtin get_unsigned_value2	106
4.167	Builtin get_variable_string	107
4.168	Builtin GetCurrentDate	107
4.169	Builtin GetCurrentDateAndTime	108
4.170	Builtin GetCurrentTime	108
4.171	Builtin iassign	108
4.172	Builtin igetval	109
4.173	Builtin IGNORE_ALL_COMM_STATUS	109
4.174	Builtin IGNORE_ALL_DEVICE_STATUS	109
4.175	Builtin IGNORE_ALL_RESPONSE_CODES	110
4.176	Builtin IGNORE_COMM_ERROR	110
4.177	Builtin IGNORE_COMM_STATUS	111
4.178	Builtin IGNORE_DEVICE_STATUS	111
4.179	Builtin IGNORE_NO_DEVICE	112
4.180	Builtin IGNORE_RESPONSE_CODE	112
4.181	Builtin int_value	113
4.182	Builtin is_NaN	113

4.183	Builtin isetval	114
4.184	Builtin isOffline.....	114
4.185	Builtin ITEM_ID	114
4.186	Builtin itoa (version A)	115
4.187	Builtin itoa (version B)	115
4.188	Builtin ivar_value.....	115
4.189	Builtin lassign.....	116
4.190	Builtin lgetval	116
4.191	Builtin ListDeleteElementAt	116
4.192	Builtin ListDeleteElementAt2	117
4.193	Builtin ListInsert	117
4.194	Builtin ListInsert2	118
4.195	Builtin log.....	118
4.196	Builtin LOG_MESSAGE	119
4.197	Builtin log10	119
4.198	Builtin log2.....	119
4.199	Builtin long_value.....	120
4.200	Builtin LongToByte	120
4.201	Builtin lsetval	121
4.202	Builtin lvar_value.....	121
4.203	Builtin Make_Time.....	121
4.204	Builtin MEMBER_ID	122
4.205	Builtin MenuDisplay.....	122
4.206	Builtin method_abort	123
4.207	Builtin nan.....	123
4.208	Builtin NaN_value	124
4.209	Builtin nanf.....	124
4.210	Builtin ObjectReference.....	125
4.211	Builtin openTransferPort.....	125
4.212	Builtin pop_abort_method.....	126
4.213	Builtin pow	126
4.214	Builtin process_abort.....	126
4.215	Builtin push_abort_method	127
4.216	Builtin put_date.....	127
4.217	Builtin put_date_value	127
4.218	Builtin put_date_value2	128
4.219	Builtin put_double	129
4.220	Builtin put_double_value	129
4.221	Builtin put_double_value2	130
4.222	Builtin put_float.....	130
4.223	Builtin put_float_value.....	131
4.224	Builtin put_float_value2	131
4.225	Builtin PUT_MESSAGE	132
4.226	Builtin put_message.....	133
4.227	Builtin put_signed.....	134
4.228	Builtin put_signed_value.....	134
4.229	Builtin put_signed_value2.....	135
4.230	Builtin put_string	135
4.231	Builtin put_string_value	136

4.232	Builtin put_string_value2	136
4.233	Builtin put_unsigned	137
4.234	Builtin put_unsigned_value	138
4.235	Builtin put_unsigned_value2	138
4.236	Builtin re_read_file	139
4.237	Builtin re_write_file	139
4.238	Builtin read_value	140
4.239	Builtin read_value2	140
4.240	Builtin ReadCommand	141
4.241	Builtin readItemFromDevice	141
4.242	Builtin remove_abort_method (version A)	142
4.243	Builtin remove_abort_method (version B)	142
4.244	Builtin remove_all_abort_methods	142
4.245	Builtin resolve_array_ref	143
4.246	Builtin resolve_array_ref2	143
4.247	Builtin resolve_block_ref	144
4.248	Builtin resolve_block_ref2	145
4.249	Builtin resolve_list_ref	145
4.250	Builtin resolve_local_ref	146
4.251	Builtin resolve_local_ref2	146
4.252	Builtin resolve_param_list_ref	147
4.253	Builtin resolve_param_ref	147
4.254	Builtin resolve_param_ref2	148
4.255	Builtin resolve_record_ref	148
4.256	Builtin resolve_record_ref2	149
4.257	Builtin ret_double_value	149
4.258	Builtin ret_double_value2	150
4.259	Builtin ret_float_value	150
4.260	Builtin ret_float_value2	151
4.261	Builtin ret_signed_value	151
4.262	Builtin ret_signed_value2	152
4.263	Builtin ret_unsigned_value	152
4.264	Builtin ret_unsigned_value2	152
4.265	Builtin retry_on_all_comm_errors	153
4.266	Builtin RETRY_ON_ALL_COMM_STATUS	153
4.267	Builtin RETRY_ON_ALL_DEVICE_STATUS	154
4.268	Builtin RETRY_ON_ALL_RESPONSE_CODES	154
4.269	Builtin retry_on_all_response_codes	155
4.270	Builtin RETRY_ON_COMM_ERROR	155
4.271	Builtin retry_on_comm_error	156
4.272	Builtin RETRY_ON_COMM_STATUS	156
4.273	Builtin RETRY_ON_DEVICE_STATUS	157
4.274	Builtin RETRY_ON_NO_DEVICE	157
4.275	Builtin RETRY_ON_RESPONSE_CODE	158
4.276	Builtin retry_on_response_code	158
4.277	Builtin round	159
4.278	Builtin save_on_exit	159
4.279	Builtin save_values	160
4.280	Builtin seconds_to_TIME_VALUE	160

4.281	Builtin seconds_to_TIME_VALUE8	160
4.282	Builtin SELECT_FROM_LIST	161
4.283	Builtin select_from_list	161
4.284	Builtin select_from_menu	162
4.285	Builtin select_from_menu2	163
4.286	Builtin send	163
4.287	Builtin send_all_values	164
4.288	Builtin send_command	164
4.289	Builtin send_command_trans	165
4.290	Builtin send_on_exit	165
4.291	Builtin send_trans	166
4.292	Builtin send_value	166
4.293	Builtin send_value2	167
4.294	Builtin SET_NUMBER_OF_RETRIES	168
4.295	Builtin sgetval	168
4.296	Builtin ShortToByte	168
4.297	Builtin sin	169
4.298	Builtin sinh	169
4.299	Builtin sqrt	169
4.300	Builtin ssetval	170
4.301	Builtin strcmp	170
4.302	Builtin strleft	170
4.303	Builtin strlen	171
4.304	Builtin strlwr	171
4.305	Builtin strmid	171
4.306	Builtin strright	172
4.307	Builtin strstr	172
4.308	Builtin strtrim	172
4.309	Builtin strupr	173
4.310	Builtin tan	173
4.311	Builtin tanh	173
4.312	Builtin Time_To_Date	174
4.313	Builtin TIME_VALUE_to_Hour	174
4.314	Builtin TIME_VALUE_to_Minute	174
4.315	Builtin TIME_VALUE_to_Second	175
4.316	Builtin TIME_VALUE_to_seconds	175
4.317	Builtin TIME_VALUE_to_string	176
4.318	Builtin timet_to_string	176
4.319	Builtin timet_to_TIME_VALUE	177
4.320	Builtin timet_to_TIME_VALUE8	177
4.321	Builtin To_Date	177
4.322	Builtin To_Date_and_Time	178
4.323	Builtin To_Time	178
4.324	Builtin To_TIME_VALUE	179
4.325	Builtin To_TIME_VALUE8	179
4.326	Builtin trunc	180
4.327	Builtin VARID	180
4.328	Builtin vassign	181
4.329	Builtin WriteCommand	181

4.330	Builtin writeltemToDevice	181
4.331	Builtin XMTR_ABORT_ON_ALL_COMM_STATUS	182
4.332	Builtin XMTR_ABORT_ON_ALL_DATA	182
4.333	Builtin XMTR_ABORT_ON_ALL_DEVICE_STATUS	183
4.334	Builtin XMTR_ABORT_ON_ALL_RESPONSE_CODES	183
4.335	Builtin XMTR_ABORT_ON_COMM_ERROR	184
4.336	Builtin XMTR_ABORT_ON_COMM_STATUS	184
4.337	Builtin XMTR_ABORT_ON_DATA	185
4.338	Builtin XMTR_ABORT_ON_DEVICE_STATUS	185
4.339	Builtin XMTR_ABORT_ON_NO_DEVICE	186
4.340	Builtin XMTR_ABORT_ON_RESPONSE_CODE	186
4.341	Builtin XMTR_IGNORE_ALL_COMM_STATUS	187
4.342	Builtin XMTR_IGNORE_ALL_DATA	187
4.343	Builtin XMTR_IGNORE_ALL_DEVICE_STATUS	188
4.344	Builtin XMTR_IGNORE_ALL_RESPONSE_CODES	188
4.345	Builtin XMTR_IGNORE_COMM_ERROR	189
4.346	Builtin XMTR_IGNORE_COMM_STATUS	189
4.347	Builtin XMTR_IGNORE_DATA	190
4.348	Builtin XMTR_IGNORE_DEVICE_STATUS	190
4.349	Builtin XMTR_IGNORE_NO_DEVICE	191
4.350	Builtin XMTR_IGNORE_RESPONSE_CODE	191
4.351	Builtin XMTR_RETRY_ON_ALL_COMM_STATUS	192
4.352	Builtin XMTR_RETRY_ON_ALL_DATA	192
4.353	Builtin XMTR_RETRY_ON_ALL_DEVICE_STATUS	193
4.354	Builtin XMTR_RETRY_ON_ALL_RESPONSE_CODE (removed)	193
4.355	Builtin XMTR_RETRY_ON_ALL_RESPONSE_CODES	193
4.356	Builtin XMTR_RETRY_ON_COMM_ERROR	194
4.357	Builtin XMTR_RETRY_ON_COMM_STATUS	194
4.358	Builtin XMTR_RETRY_ON_DATA	195
4.359	Builtin XMTR_RETRY_ON_DEVICE_STATUS	195
4.360	Builtin XMTR_RETRY_ON_NO_DEVICE	196
4.361	Builtin XMTR_RETRY_ON_RESPONSE_CODE	196
5	Builtins return codes	197
Annex A	(normative) Profiles of Builtins	199
A.1	Conventions for profiles of Builtins	199
A.2	Profiles for PROFIBUS and PROFINET	199
A.3	Profiles for FOUNDATION™ Fieldbus	208
A.4	Profiles for HART®	216
A.5	Profiles for Communication Servers	225
Table 1	– Format for the Builtins lexical element tables	23
Table 2	– Contents of the lexical element table	24
Table 3	– Usage of Builtins	24
Table 4	– User interface Builtins	25
Table 5	– Communication Builtins	26
Table 6	– Action Builtins	28
Table 7	– Builtin _ERROR	28
Table 8	– Builtin _TRACE	29

Table 9 – Builtin _WARNING.....	29
Table 10 – Builtin abort.....	30
Table 11 – Builtin abort_on_all_comm_errors.....	30
Table 12 – Builtin ABORT_ON_ALL_COMM_STATUS.....	31
Table 13 – Builtin ABORT_ON_ALL_DEVICE_STATUS.....	31
Table 14 – Builtin ABORT_ON_ALL_RESPONSE_CODES.....	32
Table 15 – Builtin abort_on_all_response_codes.....	32
Table 16 – Builtin ABORT_ON_COMM_ERROR.....	32
Table 17 – Builtin abort_on_comm_error.....	33
Table 18 – Builtin ABORT_ON_COMM_STATUS.....	34
Table 19 – Builtin ABORT_ON_DEVICE_STATUS.....	34
Table 20 – Builtin ABORT_ON_NO_DEVICE.....	35
Table 21 – Builtin ABORT_ON_RESPONSE_CODE.....	36
Table 22 – Builtin abort_on_response_code.....	36
Table 23 – Builtin abortTransferPort.....	36
Table 24 – Builtin abs.....	37
Table 25 – Builtin ACKNOWLEDGE.....	37
Table 26 – Builtin acknowledge.....	37
Table 27 – Builtin acos.....	38
Table 28 – Builtin add_abort_method.....	38
Table 29 – Builtin add_abort_method.....	39
Table 30 – Builtin AddTime.....	39
Table 31 – Builtin asin.....	40
Table 32 – Builtin assign.....	40
Table 33 – Builtin assign_double.....	40
Table 34 – Builtin assign_float.....	41
Table 35 – Builtin assign_int.....	41
Table 36 – Builtin assign_var.....	42
Table 37 – Builtin assign2.....	42
Table 38 – Builtin atan.....	43
Table 39 – Builtin atof.....	43
Table 40 – Builtin atoi.....	43
Table 41 – Builtin browseIdentity.....	44
Table 42 – Builtin BUILD_MESSAGE.....	44
Table 43 – Builtin ByteToDouble.....	44
Table 44 – Builtin ByteToFloat.....	45
Table 45 – Builtin ByteToLong.....	45
Table 46 – Builtin ByteToShort.....	46
Table 47 – Builtin cbrt.....	46
Table 48 – Builtin ceil.....	46
Table 49 – Builtin closeTransferPort.....	47
Table 50 – Builtin cos.....	47
Table 51 – Builtin cosh.....	47

Table 52 – Builtin dassign	48
Table 53 – Builtin DATE_AND_TIME_VALUE_to_string	48
Table 54 – Builtin Date_to_DayOfMonth	48
Table 55 – Builtin DATE_to_days	49
Table 56 – Builtin Date_to_Month	49
Table 57 – Builtin DATE_to_string	50
Table 58 – Builtin Date_To_Time	50
Table 59 – Builtin Date_to_Year	50
Table 60 – Builtin days_to_DATE	51
Table 61 – Builtin DELAY	51
Table 62 – Builtin delay	52
Table 63 – Builtin DELAY_TIME	52
Table 64 – Builtin delayfor	53
Table 65 – Builtin delayfor2	53
Table 66 – Builtin DICT_ID	54
Table 67 – Builtin dictionary_string	54
Table 68 – Builtin DiffTime	54
Table 69 – Builtin discard_on_exit	55
Table 70 – Builtin DISPLAY	56
Table 71 – Builtin display	56
Table 72 – Builtin display_bitenum	56
Table 73 – Builtin display_builtin_error	57
Table 74 – Builtin display_comm_error	57
Table 75 – Builtin display_comm_status	58
Table 76 – Builtin display_device_status	58
Table 77 – Builtin display_dynamics	59
Table 78 – Builtin display_dynamics2	60
Table 79 – Builtin display_message	60
Table 80 – Builtin display_message2	61
Table 81 – Builtin display_response_code	62
Table 82 – Builtin display_response_status	62
Table 83 – Builtin display_xmtr_status	63
Table 84 – Builtin DoubleToByte	63
Table 85 – Builtin drand	63
Table 86 – Builtin dseed	64
Table 87 – Builtin edit_device_value	65
Table 88 – Builtin edit_device_value2	66
Table 89 – Builtin edit_local_value	67
Table 90 – Builtin edit_local_value2	68
Table 91 – Builtin exp	68
Table 92 – Builtin ext_send_command	69
Table 93 – Builtin ext_send_command_trans	69
Table 94 – Builtin fail_on_all_comm_errors	70

Table 95 – Builtin fail_on_all_response_codes	70
Table 96 – Builtin fail_on_comm_error	71
Table 97 – Builtin fail_on_response_code.....	72
Table 98 – Builtin fassign	72
Table 99 – Builtin fGetByte	72
Table 100 – Builtin fgetval.....	73
Table 101 – Builtin float_value	73
Table 102 – Builtin FloatToByte	73
Table 103 – Builtin floor	74
Table 104 – Builtin fmod	74
Table 105 – Builtin fpclassifyd.....	75
Table 106 – Builtin From_DATE_AND_TIME_VALUE.....	75
Table 107 – Builtin From_TIME_VALUE	76
Table 108 – Builtin fsetval.....	76
Table 109 – Builtin ftoa	76
Table 110 – Builtin fvar_value	77
Table 111 – Builtin get_acknowledgement.....	77
Table 112 – Builtin get_acknowledgement2.....	78
Table 113 – Builtin get_block_instance_by_object_index.....	78
Table 114 – Builtin get_block_instance_by_tag.....	79
Table 115 – Builtin get_block_instance_count	79
Table 116 – Builtin get_comm_error.....	80
Table 117 – Builtin get_comm_error_string.....	80
Table 118 – Builtin get_date.....	81
Table 119 – Builtin get_date_lelem.....	81
Table 120 – Builtin get_date_lelem2.....	82
Table 121 – Builtin get_date_value.....	82
Table 122 – Builtin get_date_value2.....	83
Table 123 – Builtin GET_DD_REVISION	83
Table 124 – Builtin get_dds_error.....	84
Table 125 – Builtin GET_DEV_VAR_VALUE.....	84
Table 126 – Builtin get_dev_var_value	85
Table 127 – Builtin GET_DEVICE_REVISION.....	85
Table 128 – Builtin GET_DEVICE_TYPE	85
Table 129 – Builtin get_dictionary_string	86
Table 130 – Builtin get_double	86
Table 131 – Builtin get_double_lelem.....	87
Table 132 – Builtin get_double_lelem2	87
Table 133 – Builtin get_double_value	88
Table 134 – Builtin get_double_value2	88
Table 135 – Builtin get_enum_string	89
Table 136 – Builtin get_float.....	89
Table 137 – Builtin get_float_lelem.....	90

Table 138 – Builtin get_float_lelem2	90
Table 139 – Builtin get_float_value	91
Table 140 – Builtin get_float_value2	91
Table 141 – Builtin GET_LOCAL_VAR_VALUE	92
Table 142 – Builtin get_local_var_value	92
Table 143 – Builtin GET_MANUFACTURER	92
Table 144 – Builtin get_more_status	93
Table 145 – Builtin get_resolve_status	93
Table 146 – Builtin get_response_code	94
Table 147 – Builtin get_response_code_string	95
Table 148 – Builtin get_rspcode_string	95
Table 149 – Builtin get_rspcode_string_by_id	96
Table 150 – Builtin get_signed	96
Table 151 – Builtin get_signed_lelem	97
Table 152 – Builtin get_signed_lelem2	97
Table 153 – Builtin get_signed_value	98
Table 154 – Builtin get_signed_value2	98
Table 155 – Builtin get_status_code_string	99
Table 156 – Builtin get_status_string	99
Table 157 – Builtin get_stddict_string	100
Table 158 – Builtin get_string	100
Table 159 – Builtin get_string_lelem	101
Table 160 – Builtin get_string_lelem2	102
Table 161 – Builtin get_string_value	102
Table 162 – Builtin get_string_value2	103
Table 163 – Builtin GET_TICK_COUNT	103
Table 164 – Builtin get_transfer_status	104
Table 165 – Builtin get_unsigned	104
Table 166 – Builtin get_unsigned_lelem	105
Table 167 – Builtin get_unsigned_lelem2	105
Table 168 – Builtin get_unsigned_value	106
Table 169 – Builtin get_unsigned_value2	107
Table 170 – Builtin get_variable_string	107
Table 171 – Builtin GetCurrentDate	107
Table 172 – Builtin GetCurrentDateAndTime	108
Table 173 – Builtin GetCurrentTime	108
Table 174 – Builtin iassign	108
Table 175 – Builtin igetval	109
Table 176 – Builtin IGNORE_ALL_COMM_STATUS	109
Table 177 – Builtin IGNORE_ALL_DEVICE_STATUS	110
Table 178 – Builtin IGNORE_ALL_RESPONSE_CODES	110
Table 179 – Builtin IGNORE_COMM_ERROR	111
Table 180 – Builtin IGNORE_COMM_STATUS	111

Table 181 – Builtin IGNORE_DEVICE_STATUS	112
Table 182 – Builtin IGNORE_NO_DEVICE	112
Table 183 – Builtin IGNORE_RESPONSE_CODE	113
Table 184 – Builtin int_value	113
Table 185 – Builtin is_NaN	113
Table 186 – Builtin isetval	114
Table 187 – Builtin isOffline	114
Table 188 – Builtin ITEM_ID	115
Table 189 – Builtin itoa (version A)	115
Table 190 – Builtin itoa (version B)	115
Table 191 – Builtin ivar_value	116
Table 192 – Builtin lassign	116
Table 193 – Builtin lgetval	116
Table 194 – Builtin ListDeleteElementAt	117
Table 195 – Builtin ListDeleteElementAt2	117
Table 196 – Builtin ListInsert	118
Table 197 – Builtin ListInsert2	118
Table 198 – Builtin log	119
Table 199 – Builtin LOG_MESSAGE	119
Table 200 – Builtin log10	119
Table 201 – Builtin log2	120
Table 202 – Builtin long_value	120
Table 203 – Builtin LongToByte	120
Table 204 – Builtin lsetval	121
Table 205 – Builtin lvar_value	121
Table 206 – Builtin Make_Time	122
Table 207 – Builtin MEMBER_ID	122
Table 208 – Builtin MenuDisplay	123
Table 209 – Builtin method_abort	123
Table 210 – Builtin nan	124
Table 211 – Builtin NaN_value	124
Table 212 – Builtin nanf	125
Table 213 – Builtin ObjectReference	125
Table 214 – Builtin openTransferPort	126
Table 215 – Builtin pop_abort_method	126
Table 216 – Builtin pow	126
Table 217 – Builtin process_abort	127
Table 218 – Builtin push_abort_method	127
Table 219 – Builtin put_date	127
Table 220 – Builtin put_date_value	128
Table 221 – Builtin put_date_value2	129
Table 222 – Builtin put_double	129
Table 223 – Builtin put_double_value	130

Table 224 – Builtin put_double_value2	130
Table 225 – Builtin put_float.....	131
Table 226 – Builtin put_float_value.....	131
Table 227 – Builtin put_float_value2.....	132
Table 228 – Format options.....	132
Table 229 – Builtin PUT_MESSAGE	133
Table 230 – Builtin put_message.....	133
Table 231 – Builtin put_signed	134
Table 232 – Builtin put_signed_value	134
Table 233 – Builtin put_signed_value2	135
Table 234 – Builtin put_string.....	136
Table 235 – Builtin put_string_value.....	136
Table 236 – Builtin put_string_value2.....	137
Table 237 – Builtin put_unsigned	138
Table 238 – Builtin put_unsigned_value	138
Table 239 – Builtin put_unsigned_value2	139
Table 240 – Builtin re_read_file.....	139
Table 241 – Builtin re_write_file	140
Table 242 – Builtin read_value	140
Table 243 – Builtin read_value2	141
Table 244 – Builtin ReadCommand	141
Table 245 – Builtin readItemFromDevice	141
Table 246 – Builtin remove_abort_method.....	142
Table 247 – Builtin remove_abort_method.....	142
Table 248 – Builtin remove_all_abort_methods.....	143
Table 249 – Builtin resolve_array_ref.....	143
Table 250 – Builtin resolve_array_ref2	144
Table 251 – Builtin resolve_block_ref	144
Table 252 – Builtin resolve_block_ref2	145
Table 253 – Builtin resolve_list_ref.....	145
Table 254 – Builtin resolve_local_ref.....	146
Table 255 – Builtin resolve_local_ref2	146
Table 256 – Builtin resolve_param_list_ref	147
Table 257 – Builtin resolve_param_ref	147
Table 258 – Builtin resolve_param_ref2.....	148
Table 259 – Builtin resolve_record_ref	149
Table 260 – Builtin resolve_record_ref2	149
Table 261 – Builtin ret_double_value.....	150
Table 262 – Builtin ret_double_value2.....	150
Table 263 – Builtin ret_float_value	151
Table 264 – Builtin ret_float_value2	151
Table 265 – Builtin ret_signed_value.....	151
Table 266 – Builtin ret_signed_value2.....	152

Table 267 – Builtin ret_unsigned_value	152
Table 268 – Builtin ret_unsigned_value2	153
Table 269 – Builtin retry_on_all_comm_errors	153
Table 270 – Builtin RETRY_ON_ALL_COMM_STATUS	154
Table 271 – Builtin RETRY_ON_ALL_DEVICE_STATUS	154
Table 272 – Builtin RETRY_ON_ALL_RESPONSE_CODES	155
Table 273 – Builtin retry_on_all_response_codes	155
Table 274 – Builtin RETRY_ON_COMM_ERROR	156
Table 275 – Builtin retry_on_comm_error	156
Table 276 – Builtin RETRY_ON_COMM_STATUS	157
Table 277 – Builtin RETRY_ON_DEVICE_STATUS	157
Table 278 – Builtin RETRY_ON_NO_DEVICE	158
Table 279 – Builtin RETRY_ON_RESPONSE_CODE	158
Table 280 – Builtin retry_on_response_code	159
Table 281 – Builtin round	159
Table 282 – Builtin save_on_exit	160
Table 283 – Builtin save_values	160
Table 284 – Builtin seconds_to_TIME_VALUE	160
Table 285 – Builtin seconds_to_TIME_VALUE	161
Table 286 – Builtin SELECT_FROM_LIST	161
Table 287 – Builtin select_from_list	162
Table 288 – Builtin select_from_menu	162
Table 289 – Builtin select_from_menu2	163
Table 290 – Builtin send	164
Table 291 – Builtin send_all_values	164
Table 292 – Builtin send_command	165
Table 293 – Builtin send_command_trans	165
Table 294 – Builtin send_on_exit	166
Table 295 – Builtin send_trans	166
Table 296 – Builtin send_value	167
Table 297 – Builtin send_value2	167
Table 298 – Builtin SET_NUMBER_OF_RETRIES	168
Table 299 – Builtin sgetval	168
Table 300 – Builtin ShortToByte	169
Table 301 – Builtin sin	169
Table 302 – Builtin sinh	169
Table 303 – Builtin sqrt	170
Table 304 – Builtin ssetval	170
Table 305 – Builtin strcmp	170
Table 306 – Builtin strleft	171
Table 307 – Builtin strlen	171
Table 308 – Builtin strlwr	171
Table 309 – Builtin strmid	172

Table 310 – Builtin strright	172
Table 311 – Builtin strstr	172
Table 312 – Builtin strtrim	173
Table 313 – Builtin strupr	173
Table 314 – Builtin tan	173
Table 315 – Builtin tanh	174
Table 316 – Builtin Time_To_Date	174
Table 317 – Builtin TIME_VALUE_to_Hour	174
Table 318 – Builtin TIME_VALUE_to_Minute	175
Table 319 – Builtin TIME_VALUE_to_Second	175
Table 320 – Builtin TIME_VALUE_to_seconds	176
Table 321 – Builtin TIME_VALUE_to_string	176
Table 322 – Builtin timet_to_string	177
Table 323 – Builtin timet_to_TIME_VALUE	177
Table 324 – Builtin timet_to_TIME_VALUE8	177
Table 325 – Builtin To_Date	178
Table 326 – Builtin To_Date_and_Time	178
Table 327 – Builtin To_Time	179
Table 328 – Builtin To_TIME_VALUE	179
Table 329 – Builtin To_TIME_VALUE8	180
Table 330 – Builtin trunc	180
Table 331 – Builtin VARID	180
Table 332 – Builtin vassign	181
Table 333 – Builtin WriteCommand	181
Table 334 – Builtin writeItemToDevice	182
Table 335 – Builtin XMTR_ABORT_ON_ALL_COMM_STATUS	182
Table 336 – Builtin XMTR_ABORT_ON_ALL_DATA	183
Table 337 – Builtin XMTR_ABORT_ON_ALL_DEVICE_STATUS	183
Table 338 – Builtin XMTR_ABORT_ON_ALL_RESPONSE_CODES	184
Table 339 – Builtin XMTR_ABORT_ON_COMM_ERROR	184
Table 340 – Builtin XMTR_ABORT_ON_COMM_STATUS	185
Table 341 – Builtin XMTR_ABORT_ON_DATA	185
Table 342 – Builtin XMTR_ABORT_ON_DEVICE_STATUS	186
Table 343 – Builtin XMTR_ABORT_ON_NO_DEVICE	186
Table 344 – Builtin XMTR_ABORT_ON_RESPONSE_CODE	187
Table 345 – Builtin XMTR_IGNORE_ALL_COMM_STATUS	187
Table 346 – Builtin XMTR_IGNORE_ALL_DATA	188
Table 347 – Builtin XMTR_IGNORE_ALL_DEVICE_STATUS	188
Table 348 – Builtin XMTR_IGNORE_ALL_RESPONSE_CODES	189
Table 349 – Builtin XMTR_IGNORE_COMM_ERROR	189
Table 350 – Builtin XMTR_IGNORE_COMM_STATUS	190
Table 351 – Builtin XMTR_IGNORE_DATA	190
Table 352 – Builtin XMTR_IGNORE_DEVICE_STATUS	191

Table 353 – Builtin XMTR_IGNORE_NO_DEVICE	191
Table 354 – Builtin XMTR_IGNORE_RESPONSE_CODE.....	192
Table 355 – Builtin XMTR_RETRY_ON_ALL_COMM_STATUS	192
Table 356 – Builtin XMTR_RETRY_ON_ALL_DATA.....	193
Table 357 – Builtin XMTR_RETRY_ON_ALL_DEVICE_STATUS	193
Table 358 – Builtin XMTR_RETRY_ON_ALL_RESPONSE_CODES	194
Table 359 – Builtin XMTR_RETRY_ON_COMM_ERROR	194
Table 360 – Builtin XMTR_RETRY_ON_COMM_STATUS	195
Table 361 – Builtin XMTR_RETRY_ON_DATA.....	195
Table 362 – Builtin XMTR_RETRY_ON_DEVICE_STATUS	196
Table 363 – Builtin XMTR_RETRY_ON_NO_DEVICE	196
Table 364 – Builtin XMTR_RETRY_ON_RESPONSE_CODE.....	197
Table 365 – Contents of the return codes description table	197
Table 366 – Return code descriptions	198
Table 367 – Return code descriptions	198
Table A.1 – Builtin profile for PROFIBUS and PROFINET	199
Table A.2 – Builtin profile for FOUNDATION fieldbus	208
Table A.3 – Builtin profile for HART	216
Table A.4 – Builtin profile for Communication Servers.....	225

With a click to view the full PDF of IEC 61804-5:2015

IECNORM.COM

INTERNATIONAL ELECTROTECHNICAL COMMISSION

**FUNCTION BLOCKS (FB) FOR PROCESS CONTROL AND
ELECTRONIC DEVICE DESCRIPTION LANGUAGE (EDDL) –****Part 5: EDDL Builtin library****FOREWORD**

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

International Standard IEC 61804-5 has been prepared by subcommittee 65E: Devices and integration in enterprise systems, of IEC technical committee 65: Industrial-process measurement, control and automation.

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

FDIS	Report on voting
65E/450/FDIS	65E/461/RVD

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

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

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

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

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

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

Withdrawn
IECNORM.COM: Click to view the full PDF of IEC 61804-5:2015

INTRODUCTION

This part of IEC 61804 is an extension to IEC 61804-3. In earlier editions of IEC 61804-3 the EDDL Builtin library was part of the Electronic Device Description Language (EDDL) document IEC 61804-3. Because of editorial problems, the EDDL Builtin library is now moved to this separate part of IEC 61804.

IECNORM.COM: Click to view the full PDF of IEC 61804-5:2015

Withd2W

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

Part 5: EDDL Builtin library

1 Scope

This part of IEC 61804 specifies the EDDL Builtin library and provides the profiles of the various fieldbuses.

2 Normative references

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

IEC 61804-3:2015, *Function blocks (FB) for process control and Electronic Device Description Language (EDDL) – Part 3: EDDL syntax and semantics*

IEEE 754, *Standard for Binary Floating-Point Arithmetic for microprocessor systems*

3 Terms, definitions, acronyms and abbreviated terms

3.1 Terms and definitions

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

3.2 Acronyms and abbreviated terms

CP	Communication profile
CPF	Communication profile family
DD	Device Description
EDD	Electronic Device Description
EDDL	Electronic Device Description Language
HCF	HART® ¹ Communication Foundation
I/O	Input/output
ID	Identifier

¹ HART® is a registered trademark of the consortium HART Communication Foundation (HCF) (non-profit organization). This information is given for the convenience of users of this standard and does not constitute an endorsement by IEC of the trade name holder or any of its products. Compliance to this profile does not require use of the trade name. Use of the trade names requires permission from the trade name holder.

PI PROFIBUS & PROFINET ² International
 PNO PROFIBUS Nutzerorganisation

4 EDDL Builtin library

4.1 General

Clause 4 describes Builtin subroutines, which can be used in an EDD. The Builtins support user or device interactions (see Annex A). 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. Clause 4 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.

4.2 Conventions for Builtin descriptions

The following conventions apply for the Builtin definitions.

- Table 1 defines the format for the Builtins lexical element tables.
- Table 2 defines the contents of the columns of Table 1.
- EDDL basic constructs and attributes referenced in this document are written in upper case letters.

Table 1 – Format for the Builtins lexical element tables

Parameter Name	Type	Direction	Usage	Description

² PROFIBUS & PROFINET are trade names of PROFIBUS Nutzerorganisation e.V. (PNO) as part of PROFIBUS & PROFINET International. PNO is a non-profit trade organization to support the fieldbus technologies PROFIBUS & PROFINET. This information is given for the convenience of users of this standard and does not constitute an endorsement by IEC of the trade name holder or any of its products. Compliance to this profile does not require use of the tradename. Use of the trade names requires permission from the trade name holder.

Table 2 – Contents of the lexical element table

Column	Text	Meaning
Parameter name	<name>	The name of the formal parameter.
	"<return>"	The return value of the Builtin.
Type	data type	Abstract data type (basic or derived) of the formal parameter.
Direction	I	An input parameter.
	O	An output parameter is implicitly passed by reference so that the called Builtin function can modify the value. The return value of a Builtin is passed by value, if any.
	I/O	An input output parameter is implicitly passed by reference so that the called Builtin function can modify the value.
	—	There is no returned value (VOID).
Usage	m	Formal parameter is mandatory.
	o	Formal parameter is optional.
	c	Formal parameter is conditional.
	—	There is no returned value (VOID).
Description	<text>	Descriptive text for the parameter.
	—	There is no returned value (VOID).

4.3 Builtin categories

4.3.1 Overview

There are the following special categories of Builtins:

- User interface Builtins
- Communication Builtins
- Action Builtins

Restrictions apply to the usage of Builtins, depending on their category (see Table 3).

Table 3 – Usage of Builtins

Builtin usage	User interface Builtins	Communication Builtins	Action Builtins
Methods referenced in user interface menus	Allowed	Allowed	Not allowed
PRE_EDIT_ACTIONS	Allowed	Not recommended	Allowed ^a
POST_EDIT_ACTIONS	Allowed	Not recommended	Allowed ^a
REFRESH_ACTIONS	Not recommended	GRAPH and CHART: allowed Others: not recommended	Allowed ^a
PRE_READ_ACTIONS	Not recommended	Allowed	Allowed ^a
POST_READ_ACTIONS	Not recommended	Allowed	Allowed ^a
PRE_WRITE_ACTIONS	Not recommended	Allowed	Allowed ^a
POST_WRITE_ACTIONS	Not recommended	Allowed	Allowed ^a
INIT_ACTIONS	Not recommended	Allowed	Not allowed
EXIT_ACTIONS	Not recommended	Allowed	Not allowed
^a Shall only be used for actions called from a VARIABLE.			

4.3.2 User interface Builtins

User interface Builtins display information and in addition some of them require interaction from the user (see Table 4).

Table 4 – User interface Builtins

Builtin name
ACKNOWLEDGE
acknowledge
DELAY
delay
DISPLAY
display
display_bitenum
display_builtin_error
display_comm_error
display_comm_status
display_device_status
display_dynamics
display_dynamics2
display_message
display_message2
display_response_code
display_response_status
display_xmtr_status
edit_device_value
edit_device_value2
edit_local_value
edit_local_value2
get_acknowledgement2
get_acknowledgement
GET_DEV_VAR_VALUE
get_dev_var_value
GET_LOCAL_VAR_VALUE
get_local_var_value
MenuDisplay
PUT_MESSAGE
put_message
SELECT_FROM_LIST
select_from_list
select_from_menu
select_from_menu2

4.3.3 Communication Builtins

The communication Builtins allow methods to communicate with the device or to control the communication with the device (see Table 5).

Table 5 – Communication Builtins

Builtin name
abort_on_all_comm_errors
ABORT_ON_ALL_COMM_STATUS
ABORT_ON_ALL_DEVICE_STATUS
ABORT_ON_ALL_RESPONSE_CODES
abort_on_all_response_codes
ABORT_ON_COMM_ERROR
abort_on_comm_error
ABORT_ON_COMM_STATUS
ABORT_ON_DEVICE_STATUS
ABORT_ON_NO_DEVICE
ABORT_ON_RESPONSE_CODE
abort_on_response_code
closeTransferPort
ext_send_command
ext_send_command_trans
get_transfer_status
IGNORE_ALL_COMM_STATUS
IGNORE_ALL_DEVICE_STATUS
IGNORE_ALL_RESPONSE_CODES
IGNORE_COMM_ERROR
IGNORE_COMM_STATUS
IGNORE_DEVICE_STATUS
IGNORE_NO_DEVICE
IGNORE_RESPONSE_CODE
openTransferPort
read_value
read_value2
ReadCommand
readItemFromDevice
retry_on_all_comm_errors
RETRY_ON_ALL_COMM_STATUS
RETRY_ON_ALL_DEVICE_STATUS
RETRY_ON_ALL_RESPONSE_CODES
retry_on_all_response_codes
RETRY_ON_COMM_ERROR
retry_on_comm_error
RETRY_ON_COMM_STATUS
RETRY_ON_DEVICE_STATUS
RETRY_ON_NO_DEVICE

Builtin name
RETRY_ON_RESPONSE_CODE
retry_on_response_code
send
send_all_values
send_command
send_command_trans
send_on_exit
send_trans
send_value
send_value2
WriteCommand
writelnItemToDevice
XMTR_ABORT_ON_ALL_COMM_STATUS
XMTR_ABORT_ON_ALL_DATA
XMTR_ABORT_ON_ALL_DEVICE_STATUS
XMTR_ABORT_ON_ALL_RESPONSE_CODES
XMTR_ABORT_ON_COMM_ERROR
XMTR_ABORT_ON_COMM_STATUS
XMTR_ABORT_ON_DATA
XMTR_ABORT_ON_DEVICE_STATUS
XMTR_ABORT_ON_NO_DEVICE
XMTR_ABORT_ON_RESPONSE_CODE
XMTR_IGNORE_ALL_COMM_STATUS
XMTR_IGNORE_ALL_DATA
XMTR_IGNORE_ALL_DEVICE_STATUS
XMTR_IGNORE_ALL_RESPONSE_CODES
XMTR_IGNORE_COMM_ERROR
XMTR_IGNORE_COMM_STATUS
XMTR_IGNORE_DATA
XMTR_IGNORE_DEVICE_STATUS
XMTR_IGNORE_NO_DEVICE
XMTR_IGNORE_RESPONSE_CODE
XMTR_RETRY_ON_ALL_COMM_STATUS
XMTR_RETRY_ON_ALL_DATA
XMTR_RETRY_ON_ALL_DEVICE_STATUS
XMTR_RETRY_ON_ALL_RESPONSE_CODE
XMTR_RETRY_ON_ALL_RESPONSE_CODES
XMTR_RETRY_ON_COMM_ERROR
XMTR_RETRY_ON_COMM_STATUS
XMTR_RETRY_ON_DATA
XMTR_RETRY_ON_DEVICE_STATUS
XMTR_RETRY_ON_NO_DEVICE
XMTR_RETRY_ON_RESPONSE_CODE

4.3.4 Action Builtins

The action Builtins allow for accessing the value of the variable from which the action was called (see Table 6). These Builtins may not be used in other methods or in actions called from a MENU.

Table 6 – Action Builtins

Builtin name
fgetval
fsetval
get_date
get_double
get_float
get_signed
get_string
get_unsigned
igetval
isetval
put_date
put_double
put_float
put_signed
put_string
put_unsigned
sgetval
ssetval

4.4 Builtin _ERROR

Purpose

The Builtin _ERROR puts a textual information in a log. This information is put into the category error. The Builtin _TRACE and the Builtin _WARNING write information to this same log.

Any variable reference contained in the format string will be expanded to the current value. Method local variables and VARIABLES can be referenced in the message parameter. For more information, refer to the Builtin PUT_MESSAGE.

Lexical structure

_ERROR(text)

The lexical elements of the Builtin _ERROR are shown in Table 7.

Table 7 – Builtin _ERROR

Parameter name	Type	Direction	Usage	Description
text	DD_STRING	I	m	The textual information to be put into the log.
<return>	VOID	—	—	—

4.5 Builtin_TRACE

Purpose

The Builtin_TRACE puts a textual information in a log. This information is put into the category trace. The Builtin_ERROR and the Builtin_WARNING write information to this same log.

Any variable reference contained in the format string will be expanded to the current value. Method local variables and VARIABLES can be referenced in the message parameter. For more information, refer to the Builtin_PUT_MESSAGE.

Lexical structure

_TRACE(text)

The lexical elements of the Builtin_TRACE are shown in Table 8.

Table 8 – Builtin_TRACE

Parameter name	Type	Direction	Usage	Description
text	DD_STRING	l	m	The textual information to be put into the log.
<return>	VOID	—	—	—

4.6 Builtin_WARNING

Purpose

The Builtin_WARNING puts a textual information in a log. This information is put into the category warning. The Builtin_ERROR and the Builtin_TRACE write information to this same log.

Any variable reference contained in the format string will be expanded to the current value. Method local variables and VARIABLES can be referenced in the message parameter. For more information, refer to the Builtin_PUT_MESSAGE.

Lexical structure

_WARNING(text)

The lexical elements of the Builtin_WARNING are shown in Table 9.

Table 9 – Builtin_WARNING

Parameter name	Type	Direction	Usage	Description
text	DD_STRING	l	m	The textual information to be put into the log.
<return>	VOID	—	—	—

4.7 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 element of the Builtin abort is shown in Table 10.

Table 10 – Builtin abort

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.8 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.

Lexical structure

abort_on_all_comm_errors(VOID)

The lexical element of the Builtin abort_on_all_comm_errors is shown in Table 11.

Table 11 – Builtin abort_on_all_comm_errors

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.9 Builtin ABORT_ON_ALL_COMM_STATUS

Purpose

The Builtin ABORT_ON_ALL_COMM_STATUS will set all of the bits in the comm status abort mask. This will cause the system to abort the current method if the device returns any comm status value. Comm status is defined to be the first data octet returned in a transaction, when bit 7 of this octet is set.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See Builtin ABORT_ON_RESPONSE_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_COMM_STATUS(VOID)

The lexical element of the Builtin ABORT_ON_ALL_COMM_STATUS is shown in Table 12.

Table 12 – Builtin ABORT_ON_ALL_COMM_STATUS

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.10 Builtin ABORT_ON_ALL_DEVICE_STATUS

Purpose

The Builtin ABORT_ON_ALL_DEVICE_STATUS will set all of the bits in the device status abort mask. This will cause the system to abort the current method if the device returns any device status value. Device status is defined to be the second data octet returned in a transaction.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this function is used by the Builtins send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

ABORT_ON_ALL_DEVICE_STATUS (VOID)

The lexical element of the Builtin ABORT_ON_ALL_DEVICE_STATUS is shown in Table 13.

Table 13 – Builtin ABORT_ON_ALL_DEVICE_STATUS

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.11 Builtin ABORT_ON_ALL_RESPONSE_CODES

Purpose

The Builtin ABORT_ON_ALL_RESPONSE_CODES will set all of the bits in the response code abort mask. This will cause the system to abort the current method if the device returns any response code value.

The response code is defined to be the first data octet returned in a transaction, when bit 7 of this octet is 0.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this function is used by the Builtins send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

ABORT_ON_ALL_RESPONSE_CODES (VOID)

The lexical element of the Builtin ABORT_ON_ALL_RESPONSE_CODES is shown in Table 14.

Table 14 – Builtin ABORT_ON_ALL_RESPONSE_CODES

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.12 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 15.

Table 15 – Builtin abort_on_all_response_codes

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.13 Builtin ABORT_ON_COMM_ERROR

Purpose

The Builtin ABORT_ON_COMM_ERROR will set the no comm error mask in such a way that the method will be aborted if a comm error is found while sending a command.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_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_ERROR (VOID)

The lexical element of the Builtin ABORT_ON_COMM_ERROR is shown in Table 16.

Table 16 – Builtin ABORT_ON_COMM_ERROR

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.14 Builtin abort_on_comm_error

Purpose

The Builtin abort_on_comm_error causes the method to abort if it receives a specific communications error.

When a method is started, it does not abort on a specific communications error until Builtin abort_on_comm_error is called.

NOTE A communications error of zero is not considered a communications error.

This Builtin permits the method writer to select which communications errors should be handled by the method and it frees a method from having to handle a specific communications error code.

The communications error codes are defined by the consortia. Their related strings may be defined in a text dictionary.

Lexical structure

```
abort_on_comm_error(error)
```

The lexical elements of the Builtin abort_on_comm_error are shown in Table 17.

Table 17 – Builtin abort_on_comm_error

Parameter name	Type	Direction	Usage	Description
error	unsigned long	I	m	A communications error code.
<return>	long	O	m	One of the return codes specified in Table 366.

4.15 Builtin ABORT_ON_COMM_STATUS

Purpose

The Builtin ABORT_ON_COMM_STATUS will set the correct bit(s) in the comm status abort mask in such a way that the specified comm_status_value will cause the method to abort. Comm status is defined to be the first data octet returned in a transaction, when bit 7 of this octet is 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 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 18.

Table 18 – Builtin ABORT_ON_COMM_STATUS

Parameter name	Type	Direction	Usage	Description
comm_status	unsigned char	l	m	A bit mask for the communication status.
<return>	VOID	—	—	—

4.16 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 19.

Table 19 – Builtin ABORT_ON_DEVICE_STATUS

Parameter name	Type	Direction	Usage	Description
device_status	unsigned char	l	m	A bit mask for the device status.
<return>	VOID	—	—	—

4.17 Builtin ABORT_ON_NO_DEVICE

Purpose

The Builtin ABORT_ON_NO_DEVICE will set the no devices mask in such a way that the method will be aborted if no device is found while sending a command.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send command function for implementation of the masks. See ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this function is used by the Builtins send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

ABORT_ON_NO_DEVICE(VOID)

The lexical element of the Builtin ABORT_ON_NO_DEVICE is shown in Table 20.

Table 20 – Builtin ABORT_ON_NO_DEVICE

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.18 Builtin ABORT_ON_RESPONSE_CODE

Purpose

The Builtin ABORT_ON_RESPONSE_CODE will set the correct bit(s) in the response code abort mask such that the specified response code value will cause the method to abort. The response code is defined to be the first data octet returned in a transaction, when bit 7 of this octet is 0.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method.

The following is a list of all the available masks. The default values of the masks are as follows:

- response_code_abort_mask = ACCESS_RESTRICTED | CMD_NOT_IMPLEMENTED
- response_code_retry_mask = NO BITS SET
- cmd_48_response_code_abort_mask = ACCESS_RESTRICTED | CMD_NOT_IMPLEMENTED
- cmd_48_response_code_retry_mask = NO BITS SET
- device_status_abort_mask = DEVICE_MALFUNCTION
- device_status_retry_mask = NO BITS SET
- cmd_48_device_status_abort_mask = DEVICE_MALFUNCTION
- cmd_48_device_status_retry_mask = NO BITS SET
- comm_status_abort_mask = NO BITS SET
- comm_status_retry_mask = ALL BITS SET
- cmd_48_comm_status_abort_mask = NO BITS SET
- cmd_48_comm_status_retry_mask = ALL BITS SET
- no_device_abort_mask = ALL BITS SET
- comm_error_abort_mask = ALL BITS SET
- comm_error_retry_mask = NO BITS SET
- cmd_48_no_device_abort_mask = ALL BITS SET
- cmd_48_comm_error_abort_mask = ALL BITS SET
- cmd_48_comm_error_retry_mask = NO BITS SET
- cmd_48_data_abort_mask = NO BITS SET
- cmd_48_data_retry_mask = NO BITS SET

The mask affected by this function is used by the Builtins send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

ABORT_ON_RESPONSE_CODE(response_code)

The lexical elements of the Builtin ABORT_ON_RESPONSE_CODE are shown in Table 21.

Table 21 – Builtin ABORT_ON_RESPONSE_CODE

Parameter name	Type	Direction	Usage	Description
response_code	int	I	m	A bit mask for the response code.
<return>	VOID	—	—	—

4.19 Builtin abort_on_response_code

Purpose

The Builtin abort_on_response_code causes the method to abort upon receiving the specified response code.

The Builtin causes the method to abort if it receives a specific response code. This Builtin frees a method from having to handle a specific response code.

When a method is started, it will not abort on a specific response code until this Builtin is called. A response code is a value representing an application-specific error condition.

NOTE A response code value of zero is not considered an error response code. Each response code type is defined in the EDD.

Lexical structure

abort_on_response_code (code)

The lexical elements of the Builtin abort_on_response_code are shown in Table 22.

Table 22 – Builtin abort_on_response_code

Parameter name	Type	Direction	Usage	Description
code	unsigned long	I	m	A bit mask for the response code.
<return>	long	O	m	One of the return codes specified in Table 366.

4.20 Builtin abortTransferPort

Purpose

The abortTransferPort stops the Block Data Transfer indicated by the handle.

Lexical structure

abortTransferPort (handle)

The lexical elements of the Builtin abortTransferPort are shown in Table 23.

Table 23 – Builtin abortTransferPort

Parameter name	Type	Direction	Usage	Description
handle	unsigned long	I	m	The handle for the block transfer.
<return>	int	O	m	One of the return codes specified in Table 367.

4.21 Builtin abs

Purpose

The Builtin abs returns the absolute value of a floating-point value.

Lexical structure

abs (x)

The lexical elements of the Builtin abs are shown in Table 24.

Table 24 – Builtin abs

Parameter name	Type	Direction	Usage	Description
x	double	I	m	A floating-point value.
<return>	double	O	m	The absolute value.

4.22 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 25.

Table 25 – Builtin ACKNOWLEDGE

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	Message to be displayed.
<return>	int	O	m	One of the return codes specified in Table 367.

4.23 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 26.

Table 26 – Builtin acknowledge

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	Message to be displayed.
global_var_ids	array of int	I	m	An array that contains unique identifiers of VARIABLE instances.
<return>	int	O	m	One of the return codes specified in Table 367.

4.24 Builtin acos

Purpose

The Builtin acos returns the arc cosine of a floating-point value.

Lexical structure

`acos(x)`

The lexical elements of the Builtin acos are shown in Table 27.

Table 27 – Builtin acos

Parameter name	Type	Direction	Usage	Description
x	double	I	m	A floating-point value.
<return>	double	O	m	The arc cosine.

4.25 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 or when the user aborts the method.

Lexical structure

`add_abort_method(abort_method_id)`

The lexical elements of the Builtin add_abort_method are shown in Table 28.

Table 28 – Builtin add_abort_method

Parameter name	Type	Direction	Usage	Description
abort_method	reference	I	m	A reference to a METHOD.
<return>	int	O	m	One of the return codes specified in Table 367.

4.26 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_errors and abort_on_all_response_codes. Methods can also be explicitly aborted by a direct call to Builtin method_abort, or when an error occurs. The normal completion of a method is not considered an abort.

The abort method list is initially empty when a method begins execution. The abort methods are added to the back of the list and are removed and run from the front of the list in the order they appear, i.e. first-in first-out. An abort method may appear more than once in the list.

Abort methods may be shared between methods, meaning two methods may have the same abort method in their respective abort method lists.

The method_id parameter is specified by the method writer using the Builtin ITEM_ID and the name of the method. Builtins abort methods cannot call add_abort_method.

Lexical structure

add_abort_method(method_id)

The lexical elements of the Builtin add_abort_method are shown in Table 29.

Table 29 – Builtin add_abort_method

Parameter name	Type	Direction	Usage	Description
method_id	unsigned long	I	m	The ID of the method.
<return>	long	O	m	One of the return codes specified in Table 366.

4.27 Builtin AddTime

Purpose

The Builtin AddTime adds a number of seconds to a time_t value.

Lexical structure

AddTime(t0, seconds)

The lexical elements of the Builtin AddTime are shown in Table 30.

Table 30 – Builtin AddTime

Parameter name	Type	Direction	Usage	Description
t0	time_t	I	m	The initial time_t value.
seconds	double	I	m	The number of seconds to add.
<return>	time_t	O	m	The specified time_t value.

4.28 Builtin asin

Purpose

The Builtin asin returns the arc sine of a floating-point value.

Lexical structure

asin(x)

The lexical elements of the Builtin asin are shown in Table 31.

Table 31 – Builtin asin

Parameter name	Type	Direction	Usage	Description
x	double	I	m	A floating-point value.
<return>	double	O	m	The arc sine.

4.29 Builtin assign

Purpose

The Builtin assign sets the value of a VARIABLE to another VARIABLE's value. It sets the value of the destination VARIABLE equal to the value of the source VARIABLE.

NOTE The Builtin is a shorthand way of doing an ABN_GET_TYPE_VALUE followed by a put_type_value.

The parameters dst_id, src_id, dst_member_id, and src_member_id are specified using the Builtins ITEM_ID and MEMBER ID and the reference of the VARIABLE. Both the source VARIABLES and the destination VARIABLE shall have the same type.

Lexical structure

assign(dst_id, dst_member_id, src_id, src_member_id)

The lexical elements of the Builtin assign are shown in Table 32.

Table 32 – Builtin assign

Parameter name	Type	Direction	Usage	Description
dst_id	unsigned long	I	m	The item ID of the destination variable.
dst_member_id	unsigned long	I	m	The member ID of the destination variable.
src_id	unsigned long	I	m	The item ID of the source variable.
src_member_id	unsigned long	I	m	The member ID of the source variable.
<return>	long	O	m	One of the return codes specified in Table 366.

4.30 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 33.

Table 33 – Builtin assign_double

Parameter name	Type	Direction	Usage	Description
device_var	reference	I	m	The name of the variable.
new_value	double	I	m	The new value of the referenced variable.
<return>	int	O	m	One of the return codes specified in Table 367.

4.31 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 34.

Table 34 – Builtin assign_float

Parameter name	Type	Direction	Usage	Description
device_var	reference	I	m	The name of the variable.
new_value	double	I	m	The new value of the referenced variable.
<return>	int	O	m	One of the return codes specified in Table 367.

4.32 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 35.

Table 35 – Builtin assign_int

Parameter name	Type	Direction	Usage	Description
device_var	reference	I	m	The name of the variable.
new_value	int	I	m	The new value of the referenced variable.
<return>	int	O	m	One of the return codes specified in Table 367.

4.33 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 36.

Table 36 – Builtin assign_var

Parameter name	Type	Direction	Usage	Description
destination_var	reference	I	m	The name of the destination variable.
source_var	reference	I	m	The name of the source variable.
<return>	int	O	m	One of the return codes specified in Table 367.

4.34 Builtin assign2

Purpose

The Builtin assign2 sets the value of a VARIABLE to another VARIABLE's value. It sets the value of the destination VARIABLE equal to the value of the source VARIABLE.

The parameters dst_id, src_id, dst_member_id, and src_member_id are specified using the Builtins ITEM_ID and MEMBER_ID and the reference of the VARIABLE. Both the source VARIABLES and the destination VARIABLE shall have the same type.

Lexical structure

```
assign2(dst_blk_id, dst_blk_num, dst_id, dst_member_id, src_blk_id,
src_blk_num, src_id, src_member_id)
```

The lexical elements of the Builtin assign2 are shown in Table 37.

Table 37 – Builtin assign2

Parameter name	Type	Direction	Usage	Description
dst_blk_id	unsigned long	I	m	The item ID of the destination block.
dst_blk_num	unsigned long	I	m	The occurrence number of the destination block.
dst_id	unsigned long	I	m	The item ID of the destination variable.
dst_member_id	unsigned long	I	m	The member ID of the destination variable.
src_blk_id	unsigned long	I	m	The item ID of the source block.
src_blk_num	unsigned long	I	m	The occurrence number of the source block.
src_id	unsigned long	I	m	The item ID of the source variable.
src_member_id	unsigned long	I	m	The member ID of the source variable.
<return>	long	O	m	One of the return codes specified in Table 366.

4.35 Builtin atan

Purpose

The Builtin atan returns the arc tangent of a floating-point value.

Lexical structure

```
atan(x)
```

The lexical elements of the Builtin atan are shown in Table 38.

Table 38 – Builtin atan

Parameter Name	Type	Direction	Usage	Description
x	double	I	m	A floating-point value.
<return>	double	O	m	The arc tangent.

4.36 Builtin atof**Purpose**

The Builtin converts a text into a floating-point value.

Lexical structure

`atof(str)`

The lexical elements of the Builtin atof are shown in Table 39.

Table 39 – Builtin atof

Parameter name	Type	Direction	Usage	Description
str	char[]	I	m	The text to convert.
<return>	float	O	m	The floating-point value.

4.37 Builtin atoi**Purpose**

The Builtin converts a text into an integer value.

Lexical structure

`atoi(str)`

The lexical elements of the Builtin atoi are shown in Table 40.

Table 40 – Builtin atoi

Parameter name	Type	Direction	Usage	Description
str	char[]	I	m	The text to convert.
<return>	int	O	m	The integer value.

4.38 Builtin browselidentity**Purpose**

The BLOB declaration may not have a declared IDENTITY. The browselidentity function allows the user to select the BLOB using a Host Application specific mechanism.

Lexical structure

`browseIdentity(blob_item)`

The lexical elements of the Builtin browselidentity are shown in Table 41.

Table 41 – Builtin browselidentity

Parameter name	Type	Direction	Usage	Description
blob_item	DD_ITEM &	I	m	The BLOB for which the identity is browsed.
<return>	int	O	m	One of the return codes specified in Table 367.

4.39 Builtin BUILD_MESSAGE

Purpose

The Builtin BUILD_MESSAGE returns a string based on a format specification. Any variable reference contained in the format string will be expanded to the current value. Method local variables and VARIABLES can be referenced in the message parameter. For more information, refer to the Builtin PUT_MESSAGE.

Lexical structure

BUILD_MESSAGE(string)

The lexical elements of the Builtin BUILD_MESSAGE are shown in Table 42.

Table 42 – Builtin BUILD_MESSAGE

Parameter name	Type	Direction	Usage	Description
string	DD_STRING	I	m	The format specification.
<return>	DD_STRING	O	m	The formatted string.

4.40 Builtin ByteToDouble

Purpose

The Builtin ByteToDouble will return a double value from a vector of eight octets.

Lexical structure

ByteToDouble(in1, in2, in3, in4, in5, in6, in7, in8)

The lexical element of the Builtin ByteToDouble is shown in Table 43.

Table 43 – Builtin ByteToDouble

Parameter name	Type	Direction	Usage	Description
in1	unsigned char	I	m	First element of the double value (most significant byte).
in2	unsigned char	I	m	Second element of the double value.
in3	unsigned char	I	m	Third element of the double value.
in4	unsigned char	I	m	Fourth element of the double value.
in5	unsigned char	I	m	Fifth element of the double value.
in6	unsigned char	I	m	Sixth element of the double value.
in7	unsigned char	I	m	Seventh element of the double value.
in8	unsigned char	I	m	Eighth element of the double value (least significant byte).
<return>	double	O	m	A double value from the given vector of eight octets.

4.41 Builtin ByteToFloat

Purpose

The Builtin ByteToFloat will return a floating value from a vector of four octets.

Lexical structure

ByteToFloat(in1, in2, in3, in4)

The lexical elements of the Builtin ByteToFloat are shown in Table 44.

Table 44 – Builtin ByteToFloat

Parameter name	Type	Direction	Usage	Description
in1	unsigned char	I	m	First element of the float value (most significant byte).
in2	unsigned char	I	m	Second element of the float value.
in3	unsigned char	I	m	Third element of the float value.
in4	unsigned char	I	m	Fourth element of the float value (least significant byte).
<return>	float	O	m	A floating-point value from the given vector of four octets.

4.42 Builtin ByteToLong

Purpose

The Builtin ByteToLong will return a long value from a vector of four octets.

Lexical structure

ByteToLong(in1, in2, in3, in4)

The lexical elements of the Builtin ByteToLong are shown in Table 45.

Table 45 – Builtin ByteToLong

Parameter name	Type	Direction	Usage	Description
in1	unsigned char	I	m	First element of the long value (most significant byte).
in2	unsigned char	I	m	Second element of the long value.
in3	unsigned char	I	m	Third element of the long value.
in4	unsigned char	I	m	Fourth element of the long value (least significant byte).
<return>	long	O	m	A long value from the given vector of four octets.

4.43 Builtin ByteToShort

Purpose

The Builtin ByteToShort will return a short value from a vector of two octets.

Lexical structure

ByteToShort(in1, in2)

The lexical elements of the Builtin ByteToShort are shown in Table 46.

Table 46 – Builtin ByteToShort

Parameter name	Type	Direction	Usage	Description
in1	unsigned char	I	m	First element of the short value (most significant byte).
in2	unsigned char	I	m	Second element of the short value (least significant byte).
<return>	short	O	m	A short value from the given vector of two octets.

4.44 Builtin cbrt

Purpose

The Builtin cbrt returns the cubic root of a floating-point value.

Lexical structure

cbrt(x)

The lexical elements of the Builtin cbrt are shown in Table 47.

Table 47 – Builtin cbrt

Parameter name	Type	Direction	Usage	Description
x	double	I	m	A floating-point value.
<return>	double	O	m	The cubic root.

4.45 Builtin ceil

Purpose

The Builtin ceil returns the smallest integral value not less than a specified floating-point value.

Lexical structure

ceil(x)

The lexical elements of the Builtin ceil are shown in Table 48.

Table 48 – Builtin ceil

Parameter name	Type	Direction	Usage	Description
x	double	I	m	A floating-point value.
<return>	double	O	m	The ceiling value.

4.46 Builtin closeTransferPort

Purpose

The closeTransferPort function closes the block transfer indicated by the handle.

Lexical structure

```
closeTransferPort(handle)
```

The lexical elements of the Builtin closeTransferPort are shown in Table 49.

Table 49 – Builtin closeTransferPort

Parameter name	Type	Direction	Usage	Description
handle	unsigned long	I	m	The handle for the block transfer.
<return>	int	O	m	One of the return codes specified in Table 367.

4.47 Builtin cos**Purpose**

The Builtin cos returns the cosine of a floating-point value.

Lexical structure

```
cos(x)
```

The lexical elements of the Builtin cos are shown in Table 50.

Table 50 – Builtin cos

Parameter name	Type	Direction	Usage	Description
x	double	I	m	A floating-point value.
<return>	double	O	m	The cosine.

4.48 Builtin cosh**Purpose**

The Builtin cosh returns the hyperbolic cosine of a floating-point value.

Lexical structure

```
cosh(x)
```

The lexical elements of the Builtin cosh are shown in Table 51.

Table 51 – Builtin cosh

Parameter name	Type	Direction	Usage	Description
x	double	I	m	A floating-point value.
<return>	double	O	m	The hyperbolic cosine.

4.49 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 52.

Table 52 – Builtin dassign

Parameter name	Type	Direction	Usage	Description
device_var	reference	I	m	The name of the variable.
new_value	double	I	m	The new value of the referenced variable.
<return>	int	O	m	One of the return codes specified in Table 367.

4.50 Builtin DATE_AND_TIME_VALUE_to_string

Purpose

The Builtin DATE_AND_TIME_VALUE_to_string creates a string representation of a DATE and a TIME_VALUE(4). The programming language-C strftime function defines the structure of the format string.

Lexical structure

DATE_AND_TIME_VALUE_to_string(string, format, date, time_value)

The lexical elements of the Builtin DATE_AND_TIME_VALUE_to_string are shown in Table 53.

Table 53 – Builtin DATE_AND_TIME_VALUE_to_string

Parameter name	Type	Direction	Usage	Description
string	DD_STRING	I/O	m	The string into which the formatted DATE and TIME_VALUE are copied.
format	DD_STRING		m	How the DATE and TIME_VALUE are to be formatted.
date	long	I	m	A long value containing the DATE to be formatted.
time_value	unsigned long	I	M	The TIME_VALUE to be formatted.
<return>	int	O	m	The number of character actually copied or -1 if an error occurs.

4.51 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 54.

Table 54 – Builtin Date_to_DayOfMonth

Parameter name	Type	Direction	Usage	Description
date	long	I	m	A long value containing the DATE.
<return>	int	O	m	The day of month.

4.52 Builtin DATE_to_days

Purpose

The Builtin DATE_to_days calculates the number of days between two dates.

Lexical structure

DATE_to_days(date1, date0)

The lexical elements of the Builtin DATE_to_days are shown in Table 55.

Table 55 – Builtin DATE_to_days

Parameter name	Type	Direction	Usage	Description
date1	long	I	m	The more recent date.
date0	long	I	m	The less recent date.
<return>	long	O	m	The number of days between date1 and date0.

4.53 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 56.

Table 56 – Builtin Date_to_Month

Parameter name	Type	Direction	Usage	Description
date	long	I	m	A long value containing the DATE.
<return>	int	O	m	The month.

4.54 Builtin DATE_to_string

Purpose

The Builtin DATE_to_string creates a string representation of a DATE. The programming language-C strftime function defines the structure of the format string.

Lexical structure

DATE_to_string(string, format, date)

The lexical elements of the Builtin DATE_to_string are shown in Table 57.

Table 57 – Builtin DATE_to_string

Parameter name	Type	Direction	Usage	Description
string	DD_STRING	I	m	The string into which the formatted DATE is copied.
format	DD_STRING	I	m	How the DATE is to be formatted.
date	long	I	m	A long value containing the DATE.
<return>	int	O	m	The number of character actually copied or -1 if an error occurs.

4.55 Builtin Date_To_Time

Purpose

The Builtin Date_To_Time creates a time_t from a date value. The time of day is set to 00:00:00.

Lexical structure

Date_To_Time(date)

The lexical elements of the Builtin Date_To_Time are shown in Table 58.

Table 58 – Builtin Date_To_Time

Parameter name	Type	Direction	Usage	Description
date	long	I	m	A long value containing the DATE.
<return>	time_t	O	m	A time_t value.

4.56 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 59.

Table 59 – Builtin Date_to_Year

Parameter name	Type	Direction	Usage	Description
date	long	I	m	A long value containing the DATE.
<return>	int	O	m	The year (4 digits).

4.57 Builtin days_to_DATE

Purpose

The Builtin days_to_DATE calculates a future date by adding a number of days to a DATE.

Lexical structure

days_to_DATE(days, date)

The lexical elements of the Builtin days_to_DATE are shown in Table 60.

Table 60 – Builtin days_to_DATE

Parameter name	Type	Direction	Usage	Description
days	long	I	m	The number of days to add to the DATE.
date	long	I	m	A long value containing the initial DATE.
<return>	long	O	m	A long value containing the DATE.

4.58 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 the 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 61.

Table 61 – Builtin DELAY

Parameter name	Type	Direction	Usage	Description
delay_time	unsigned integer	I	m	The delay in number of seconds.
prompt	char[]	I	m	A message to be displayed.
<return>	VOID	—	—	—

4.59 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 the 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 62.

Table 62 – Builtin delay

Parameter name	Type	Direction	Usage	Description
delay_time	unsigned integer	I	m	The delay in number of seconds.
prompt	char[]	I	m	A message to be displayed.
global_var_ids	array of int	I	m	An array which contains unique identifiers of VARIABLE instances.
<return>	VOID	—	—	—

4.60 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 63.

Table 63 – Builtin DELAY_TIME

Parameter name	Type	Direction	Usage	Description
delay_time	unsigned integer	I	m	The delay in number of seconds.
<return>	VOID	—	—	—

4.61 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 64.

Table 64 – Builtin delayfor

Parameter name	Type	Direction	Usage	Description
duration	long	I	m	The delay in number of seconds.
prompt	char[]	I	m	A message to be displayed.
ids	array of unsigned long	I	m	The item IDs of a VARIABLE instance used in the prompt message.
indices	array of unsigned long	I	m	The member IDs or array indices of the VARIABLES used in the prompt message. Set to zero if there is no member ID or array index.
id_count	long	I	m	The number of array elements in the arrays for IDs and indices. Set to zero if the arrays for IDs and indices are NULL.
<return>	long	O	m	One of the return codes specified in Table 366.

4.62 Builtin delayfor2

Purpose

The Builtin delayfor2 prints a string on the application display and waits for the allotted amount of time to expire. VARIABLE values may be embedded in the message string with formatting information.

Lexical structure

delayfor2(duration, prompt, blk_ids, blk_nums, ids, indices, id_count)

The lexical elements of the Builtin delayfor2 are shown in Table 65.

Table 65 – Builtin delayfor2

Parameter name	Type	Direction	Usage	Description
duration	long	I	m	The delay in number of seconds.
prompt	char[]	I	m	A message to be displayed.
blk_ids	array of unsigned long	I	m	The item IDs of the BLOCK_A instances used in the prompt message.
blk_nums	array of unsigned long	I	m	The occurrence numbers of the BLOCK_A instances used in the prompt message.
ids	array of unsigned long	I	m	The item IDs of a VARIABLE instance used in the prompt message.
indices	array of unsigned long	I	m	The member IDs or array indices of the VARIABLES used in the prompt message. Set to zero if there is no member ID or array index.
id_count	long	I	m	The number of array elements in the arrays for IDs and indices. Set to zero if the arrays for IDs and indices are NULL.
<return>	long	O	m	One of the return codes specified in Table 366.

4.63 Builtin DICT_ID

Purpose

The Builtin DICT_ID can be used to specify a text dictionary ID for calls to get_stdidict_string. It converts a text dictionary name into a text dictionary ID.

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 66.

Table 66 – Builtin `DICT_ID`

Parameter name	Type	Direction	Usage	Description
name	reference	l	m	The name of the dictionary string.
<return>	long	O	m	The dictionary ID for the named string.

4.64 Builtin `dictionary_string`

Purpose

The Builtin `dictionary_string` returns the dictionary string associated with the given name in the current language. If the string is not available in the current language, the English string is returned. If the string is not found in the dictionary the function returns “ERROR: Dict String Not Found”. The parameter `dict_string_name` is the handle of the string as it appears in the dictionary.

Lexical structure

`dictionary_string (dict_string_name)`

The lexical elements of the Builtin `dictionary_string` are shown in Table 67.

Table 67 – Builtin `dictionary_string`

Parameter name	Type	Direction	Usage	Description
dict_string_name	reference	l	m	Handle of the string as it appears in the dictionary.
<return>	DD_STRING	o	m	The dictionary string.

4.65 Builtin `DiffTime`

Purpose

The Builtin `DiffTime` calculates the difference between two calendar times.

Lexical Structure

`DiffTime (t1, t0)`

The lexical elements of the Builtin `DiffTime` are shown in Table 68.

Table 68 – Builtin `DiffTime`

Parameter name	Type	Direction	Usage	Description
t1	time_t	l	m	The more recent time.
t0	time_t	l	m	The less recent time.
<return>	double	o	m	The difference between t1 and t0 in seconds.

4.66 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 cache, which have not already been sent to the device. It causes the method to discard any changes to VARIABLE values in the cache 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 cache (rather than send them to the device), call the Builtin save_on_exit.

After the method termination, the changed values in the cache 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 69.

Table 69 – Builtin discard_on_exit

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.67 Builtin DISPLAY

Purpose

The Builtin DISPLAY displays the specified message on the screen, continuously updating the dynamic variable values used in the string (see PUT_MESSAGE for syntax). The variable value will continue to be updated until operator acknowledgment is made.

Lexical structure

DISPLAY(prompt)

The lexical elements of the Builtin DISPLAY are shown in Table 70.

Table 70 – Builtin DISPLAY

Parameter name	Type	Direction	Usage	Description
prompt	DD_STRING	I	m	A message to be displayed.
<return>	VOID	—	—	—

4.68 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 71.

Table 71 – Builtin display

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	A message to be displayed.
global_var_ids	array of int	I	m	An array which contains unique identifiers of VARIABLE instances.
<return>	VOID	—	—	—

4.69 Builtin display_bitenum

Purpose

The Builtin `display_bitenum` will display a textual representation of a BIT_ENUMERATED VARIABLE and wait for acknowledgment from the user.

NOTE 1 Previously displayed messages are not guaranteed to remain on the screen.

NOTE 2 Some EDD applications will clear the display upon each call to this function.

Lexical structure

`display_bitenum(bitenum_variable)`

The lexical elements of the Builtin `display_bitenum` are shown in Table 72.

Table 72 – Builtin display_bitenum

Parameter name	Type	Direction	Usage	Description
bitenum_variable	Reference	I	m	The BIT_ENUMERATED VARIABLE that shall be displayed.
<return>	VOID	—	—	—

4.70 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.

This Builtin provides a means of debugging calls to Builtin routines, which return 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.

The Builtin error codes may be defined in a separate file.

Lexical structure

display_builtin_error(error)

The lexical elements of the Builtin display_builtin_error are shown in Table 73.

Table 73 – Builtin display_builtin_error

Parameter name	Type	Direction	Usage	Description
error	long	I	m	A code for a Builtin error.
<return>	long	O	m	One of the return codes specified in Table 366.

4.71 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 74.

Table 74 – Builtin display_comm_error

Parameter name	Type	Direction	Usage	Description
error	unsigned long	I	m	A code for a communication error.
<return>	long	O	m	One of the return codes specified in Table 366.

4.72 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 75.

Table 75 – Builtin display_comm_status

Parameter name	Type	Direction	Usage	Description
comm_status_value	int	l	m	A code for a communication error.
<return>	VOID	—	—	—

4.73 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 76.

Table 76 – Builtin display_device_status

Parameter name	Type	Direction	Usage	Description
device_status_value	int	l	m	A code for the device status.
<return>	VOID	—	—	—

4.74 Builtin display_dynamics

Purpose

The Builtin display_dynamics continuously displays a message on the application's output device. The Builtin updates the values specified in the message until user acknowledgement is made.

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 77.

Table 77 – Builtin `display_dynamics`

Parameter name	Type	Direction	Usage	Description
prompt	char[]	l	m	A message to be displayed.
ids	array of unsigned long	l	m	The item IDs of the VARIABLES used in the prompt message.
indices	array of unsigned long	l	m	The member IDs or array index of the VARIABLES used in the prompt message. Set to zero if there is no member ID or array index.
id_count	long	l	m	The number of array elements in the arrays for IDs and indices. Set to zero if the arrays for IDs and indices are NULL.
<return>	long	O	m	One of the return codes specified in Table 366.

4.75 Builtin `display_dynamics2`**Purpose**

The Builtin `display_dynamics2` continuously displays a message on the application's 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. (The application code scans for the acknowledgment; the Builtin waits for the application code to receive the user intervention, checks it, and returns control to the Builtin.)

VARIABLE values may be embedded in the message string with formatting information. This call can be made for device VARIABLES or local method VARIABLES. The primary purpose of the call is to display device value information that the user wants to observe over a period of time.

Lexical structure

```
display_dynamics2(prompt, blk_ids, blk_nums, ids, indices, id_count)
```

The lexical elements of the Builtin `display_dynamics2` are shown in Table 78.

Table 78 – Builtin display_dynamics2

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	A message to be displayed.
blk_ids	array of unsigned long	I	m	The item IDs of the BLOCK_A instances used in the prompt message.
blk_nums	array of unsigned long	I	m	The occurrence numbers of the BLOCK_A instances used in the prompt message.
ids	array of unsigned long	I	m	The item IDs of the VARIABLEs used in the prompt message.
indices	array of unsigned long	I	m	The member IDs or array index of the VARIABLEs used in the prompt message. Set to zero if there is no member ID or array index.
id_count	long	I	m	The number of array elements in the arrays for ID s and indices. Set to zero if the arrays for ID s and indices are NULL.
<return>	long	O	m	One of the return codes specified in Table 366.

4.76 Builtin display_message

Purpose

The Builtin display_message displays a message on the application's 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 79.

Table 79 – Builtin display_message

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	A message to be displayed.
ids	array of unsigned long	I	m	The item IDs of the VARIABLEs used in the prompt message.
indices	array of unsigned long	I	m	The member IDs or array index of the VARIABLEs used in the prompt message. Set to zero if there is no member ID or array index.
id_count	long	I	m	The number of array elements in the arrays for IDs and indices. Set to zero if the arrays for IDs and indices are NULL.
<return>	long	O	m	One of the return codes specified in Table 366.

4.77 Builtin display_message2

Purpose

The Builtin display_message2 displays a message on the application's output device. It returns control to the method without requiring user acknowledgment.

Device, local method VARIABLE values, and labels for device VARIABLES may be embedded in the prompt string with formatting information. This call is used to inform the user of system status changes that require no action.

Lexical structure

display_message2(prompt, blk_ids, blk_nums, ids, indices, id_count)

The lexical elements of the Builtin display_message2 are shown in Table 80

Table 80 – Builtin display_message2

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	A message to be displayed.
blk_ids	array of unsigned long	I	m	The item IDs of the BLOCK_A instances used in the prompt message.
blk_nums	array of unsigned long	I	m	The occurrence numbers of the BLOCK_A instances used in the prompt message.
ids	array of unsigned long	I	m	The item IDs of the VARIABLES used in the prompt message.
indices	array of unsigned long	I	m	The member IDs or array index of the VARIABLES used in the prompt message. Set to zero if there is no member ID or array index.
id_count	long	I	m	The number of array elements in the arrays for IDs and indices. Set to zero if the arrays for IDs and indices are NULL.
<return>	long	O	m	One of the return codes specified in Table 366.

4.78 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 81.

Table 81 – Builtin `display_response_code`

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	m	The item ID of the item which generated the response code.
member_id	unsigned long	I	m	The member ID of the item which generated the response code.
code	unsigned long	I	m	The response code as received from Builtin <code>get_response_code</code> .
<return>	long	O	m	One of the return codes specified in Table 366.

4.79 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 82.

Table 82 – Builtin `display_response_status`

Parameter name	Type	Direction	Usage	Description
command_num	int	I	m	The NUMBER of the COMMAND from which the response status has been received.
response_code_value	int	I	m	The response code to display in a prompt.
<return>	VOID	—	—	—

4.80 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 83.

Table 83 – Builtin display_xmtr_status

Parameter name	Type	Direction	Usage	Description
xmtr_variable	int	I	m	A reference to a VARIABLE instance.
response_code_value	int	I	m	A response code value.
<return>	VOID	—	—	—

4.81 Builtin DoubleToByte

Purpose

The Builtin DoubleToByte will provide a vector of unsigned char values representing the particular elements of a double value.

Lexical structure

DoubleToByte(in, out1, out2, out3, out4, out5, out6, out7, out8)

The lexical elements of the Builtin DoubleToByte are shown in Table 84.

Table 84 – Builtin DoubleToByte

Parameter name	Type	Direction	Usage	Description
in	double	I	m	The double value to convert.
out1	unsigned char	O	m	First element of the double value (most significant byte).
out2	unsigned char	O	m	Second element of the double value.
out3	unsigned char	O	m	Third element of the double value.
out4	unsigned char	O	m	Fourth element of the double value.
out5	unsigned char	O	m	Fifth element of the double value.
out6	unsigned char	O	m	Sixth element of the double value.
out7	unsigned char	O	m	Seventh element of the double value.
out8	unsigned char	O	m	Eighth element of the double value (least significant byte).
<return>	VOID	—	—	—

4.82 Builtin drand

Purpose

The Builtin drand() returns random double precision values uniformly distributed across the range from 0 to 1 .

Lexical structure

double drand()

The lexical element of the Builtin drand is shown in Table 85.

Table 85 – Builtin drand

Parameter name	Type	Direction	Usage	Description
<return>	double	O	m	Returns random double precision values uniformly distributed across the range from 0 to 1.

4.83 Builtin dseed

Purpose

The Builtin dseed() takes a double in the range 0 to 1 as the seed for the pseudo-random number generator. For a given host implementation, if dseed is called with the same seed value the same pseudo-random number sequence shall be generated.

Lexical structure

```
void dseed(double seed)
```

The lexical elements of the Builtin dseed are shown in Table 86.

Table 86 – Builtin dseed

Parameter name	Type	Direction	Usage	Description
seed	double	l	m	A double in the range 0 to 1 as the seed for the pseudo-random number generator.
<return>	VOID	—	—	—

4.84 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.

A device value may reside in the cache 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 87.

Table 87 – Builtin edit_device_value

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	A message to be displayed.
ids	array of unsigned long	I	m	The item IDs of the VARIABLES used in the prompt message.
indices	array of unsigned long	I	m	The member IDs or array index of the VARIABLES used in the prompt message. Set to zero if there is no member ID or array index.
id_count	long	I	m	The number of array elements in the arrays for IDs and indices. Set to zero if the arrays for IDs and indices are NULL.
param_id	unsigned long	I	m	The item ID of the VARIABLE to be modified.
param_member_id	unsigned long	I	m	The member ID of the VARIABLE to be modified.
<return>	long	O	m	One of the return codes specified in Table 366.

4.85 Builtin edit_device_value2

Purpose

The Builtin edit_device_value2 displays a prompt with a device value specified by the param_id and param_member_id. The user can edit the displayed value.

A device value may reside in the cache of a caching EDD application, and the value may not be written to the device until a method terminates or the value is explicitly sent to the device by the method.

VARIABLE values may be embedded in the message string with formatting information. The application edits and verifies the values entered for the device value. This editing is based on VARIABLE arithmetic types defined in the original EDD for each VARIABLE. The EDDL attributes associated with VARIABLES may include:

- HANDLING
- DISPLAY_FORMAT
- EDIT_FORMAT
- MIN_VALUE
- MAX_VALUE

Lexical structure

```
edit_device_value2(prompt, blk_ids, blk_nums, ids, indices, id_count,
blk_id, blk_num, param_id, param_member_id)
```

The lexical elements of the Builtin edit_device_value2 are shown in Table 88.

Table 88 – Builtin edit_device_value2

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	A message to be displayed.
blk_ids	array of unsigned long	I	m	The item IDs of the BLOCK_A instances used in the prompt message.
blk_nums	array of unsigned long	I	m	The occurrence numbers of the BLOCK_A instances used in the prompt message.
ids	array of unsigned long	I	m	The item IDs of the VARIABLES used in the prompt message.
indices	array of unsigned long	I	m	The member IDs or array index of the VARIABLES used in the prompt message. Set to zero if there is no member ID or array index.
id_count	long	I	m	The number of array elements in the arrays for IDs and indices. Set to zero if the arrays for IDs and indices are NULL.
blk_id	unsigned long	I	m	The item ID of the BLOCK_A instance to be modified.
blk_num	unsigned long	I	m	The occurrence number of the BLOCK_A instance to be modified.
param_id	unsigned long	I	m	The item ID of the VARIABLE to be modified.
param_member_id	unsigned long	I	m	The member ID of the VARIABLE to be modified.
<return>	long	O	m	One of the return codes specified in Table 366.

4.86 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 89.

Table 89 – Builtin edit_local_value

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	A message to be displayed.
ids	array of unsigned long	I	m	The item IDs of the VARIABLEs used in the prompt message.
indices	array of unsigned long	I	m	The member IDs or array index of the VARIABLEs used in the prompt message. Set to zero if there is no member ID or array index.
id_count	long	I	m	The number of array elements in the arrays for IDs and indices. Set to zero if the arrays for IDs and indices are NULL.
local_var	char[]	I	m	The name of the local VARIABLE that may be modified.
<return>	long	O	m	One of the return codes specified in Table 366.

4.87 Builtin edit_local_value2**Purpose**

The Builtin edit_local_value2 displays a prompt with the local method VARIABLE name specified by the parameter local_var. The local_var parameter is a null-terminated C string of characters. The user can edit the displayed value of the local method VARIABLE.

Variable values may be embedded in the message string with formatting information. The EDD application is responsible for preventing invalid values from being returned by performing checks based on VARIABLE type only. The method shall perform range checking on returned values.

Lexical structure

```
edit_local_value2(prompt, blk_ids, blk_nums, ids, indices, id_count,
local_var)
```

The lexical elements of the Builtin edit_local_value2 are shown in Table 90.

Table 90 – Builtin edit_local_value2

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	A message to be displayed.
blk_ids	array of unsigned long	I	m	The item IDs of the BLOCK_A instances used in the prompt message.
blk_nums	array of unsigned long	I	m	The occurrence numbers of the BLOCK_A instances used in the prompt message.
ids	array of unsigned long	I	m	The item IDs of the VARIABLES used in the prompt message.
indices	array of unsigned long	I	m	The member IDs or array index of the VARIABLES used in the prompt message. Set to zero if there is no member ID or array index.
id_count	long	I	m	The number of array elements in the arrays for IDs and indices. Set to zero if the arrays for IDs and indices are NULL.
local_var	char[]	I	m	The name of the local VARIABLE that may be modified.
<return>	long	O	m	One of the return codes specified in Table 366.

4.88 Builtin exp**Purpose**

The Builtin exp returns the exponential of a floating-point value.

Lexical structure

exp (x)

The lexical elements of the Builtin exp are shown in Table 91.

Table 91 – Builtin exp

Parameter name	Type	Direction	Usage	Description
x	double	I	m	A floating-point value.
<return>	double	O	m	The exponential value.

4.89 Builtin ext_send_command**Purpose**

The Builtin ext_send_command 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 92.

Table 92 – Builtin `ext_send_command`

Parameter name	Type	Direction	Usage	Description
cmd_number	int	I	m	The COMMAND number.
cmd_status	char[]	I	m	The COMMAND status.
more_data_status	char[]	I	m	Specifies if more status data is available.
more_data_info	char[]	I	m	Specifies if more info is available.
<return>	int	O	m	One of the return codes specified in Table 367.

4.90 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 93.

Table 93 – Builtin `ext_send_command_trans`

Parameter name	Type	Direction	Usage	Description
cmd_number	int	I	m	The COMMAND number.
transaction	int	I	m	The number of the TRANSACTION.
cmd_status	char[]	I	m	The COMMAND status.
more_data_status	char[]	I	m	Specifies if more status data is available.
more_data_info	char[]	I	m	Specifies if more info is available.
<return>	int	O	m	One of the return codes specified in Table 367.

4.91 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 94.

Table 94 – Builtin fail_on_all_comm_errors

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.92 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 95.

Table 95 – Builtin fail_on_all_response_codes

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.93 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 96.

Table 96 – Builtin fail_on_comm_error

Parameter name	Type	Direction	Usage	Description
error	unsigned long	I	m	A communication error code.
<return>	long	O	m	One of the return codes specified in Table 366.

4.94 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 97.

Table 97 – Builtin fail_on_response_code

Parameter name	Type	Direction	Usage	Description
code	unsigned long	I	m	A response code.
<return>	long	O	m	One of the return codes specified in Table 366.

4.95 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 98.

Table 98 – Builtin fassign

Parameter name	Type	Direction	Usage	Description
device_var	reference	I	m	The name of the VARIABLE.
new_value	double	I	m	The new value of the referenced VARIABLE.
<return>	int	O	m	One of the return codes specified in Table 367.

4.96 Builtin fGetByte

Purpose

The Builtin fGetByte allows access to BLOB contents for validation purposes only.

Lexical structure

```
fGetByte(blob_item, position, value_read)
```

The lexical elements of the Builtin fGetByte are shown in Table 99.

Table 99 – Builtin fGetByte

Parameter name	Type	Direction	Usage	Description
blob_item	DD_ITEM &	I	m	The BLOB from which the byte shall be retrieved.
position	unsigned long long	I	m	The position relative to the start from which to read.
value_read	unsigned char &	O	m	Value from the BLOB.
<return>	int	O	m	One of the return codes specified in Table 367.

4.97 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 100.

Table 100 – Builtin fgetval

Parameter name	Type	Direction	Usage	Description
<return>	double	O	m	The value of the variable.

4.98 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 101.

Table 101 – Builtin float_value

Parameter name	Type	Direction	Usage	Description
source_var_name	reference	I	m	The name of a VARIABLE reference.
<return>	double	O	m	The value of the variable.

4.99 Builtin FloatToByte**Purpose**

The Builtin FloatToByte will provide a vector of unsigned char values representing the particular elements of a float value.

Lexical structure

```
FloatToByte(in, out1, out2, out3, out4)
```

The lexical element of the Builtin FloatToByte is shown in Table 102.

Table 102 – Builtin FloatToByte

Parameter name	Type	Direction	Usage	Description
in	float	I	m	The float value to convert.
out1	unsigned char	O	m	First element of the float value (most significant byte).
out2	unsigned char	O	m	Second element of the float value.
out3	unsigned char	O	m	Third element of the float value.
out4	unsigned char	O	m	Fourth element of the float value (least significant byte).
<return>	VOID	—	—	—

4.100 Builtin floor

Purpose

The Builtin floor returns the largest integral value not greater than a specified floating-point value.

Lexical structure

`floor(x)`

The lexical elements of the Builtin floor are shown in Table 103.

Table 103 – Builtin floor

Parameter name	Type	Direction	Usage	Description
x	double	I	m	A floating-point value.
<return>	double	O	m	The floor value.

4.101 Builtin fmod

Purpose

The Builtin fmod returns the floating-point remainder of the division of two floating-point values.

Lexical structure

`fmod(x, y)`

The lexical elements of the Builtin fmod are shown in Table 104.

Table 104 – Builtin fmod

Parameter name	Type	Direction	Usage	Description
x	double	I	m	The numerator.
y	double	I	m	The denominator.
<return>	double	O	m	The remainder.

4.102 Builtin fpclassify

Purpose

The Builtin fpclassify classifies a floating-point value as either not a number, infinite, zero, subnormal, or normal.

NOTE This Builtin is implemented as a macro that generates a call to either `fpclassifyf` or `fpclassifyd` depending on whether its input parameter is a float or double, respectively.

Lexical Structure

`fpclassify(value)`

The lexical elements of the Builtin fpclassify are shown in Table 105.

Table 105 – Builtin fpclassifyd

Parameter name	Type	Direction	Usage	Description
value	float or double	l	m	The floating-point value to be classified.
<return>	int	o	m	FP_NAN when the parameter value is 'Not a Number'. FP_INFINITE when the parameter value is either plus or minus infinity. FP_ZERO when the parameter value is positive or negative zero. FP_SUBNORMAL when the parameter value is denormalized. Denormalized numbers have values that are too small (i.e., values very close to zero) to be represented otherwise. FP_NORMAL is returned for all other parameter values .

4.103 Builtin From_DATE_AND_TIME_VALUE**Purpose**

The Builtin From_DATE_AND_TIME_VALUE creates a time_t from a DATE and a TIME_VALUE(4).

Lexical structure

From_DATE_AND_TIME_VALUE(date, time_value)

The lexical elements of the Builtin From_DATE_AND_TIME_VALUE are shown in Table 106.

Table 106 – Builtin From_DATE_AND_TIME_VALUE

Parameter name	Type	Direction	Usage	Description
date	long	l	m	The DATE value.
time_value	unsigned long	l	m	The TIME_VALUE(4) value.
<return>	time_t	o	m	The time_t that was created or -1 if an invalid parameter is detected.

4.104 Builtin From_TIME_VALUE**Purpose**

The Builtin From_TIME_VALUE creates a time_t from a TIME_VALUE(4) and the current date (midnight) or from a TIME_VALUE(8).

Lexical structure

From_TIME_VALUE(time_value)

The lexical elements of the Builtin From_TIME_VALUE are shown in Table 107.

Table 107 – Builtin From_TIME_VALUE

Parameter name	Type	Direction	Usage	Description
time_value	unsigned long or unsigned long long	I	M	The TIME_VALUE value. A TIME_VALUE(8) is passed in as an unsigned long long and is an absolute time. A TIME_VALUE(4) is passed in as an unsigned long and is a time of day.
<return>	time_t	O	m	The time_t that was created or -1 if an invalid parameter is detected.

4.105 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 108.

Table 108 – Builtin fsetval

Parameter name	Type	Direction	Usage	Description
value	double	I	m	The new value for the VARIABLE.
<return>	double	O	m	The value of the variable.

4.106 Builtin ftoa**Purpose**

The Builtin converts a floating-point value into a text string.

Lexical structure

ftoa(f)

The lexical elements of the Builtin ftoa are shown in Table 109.

Table 109 – Builtin ftoa

Parameter name	Type	Direction	Usage	Description
f	float	I	m	A floating-point value to convert to a text.
<return>	char[]	O	m	The floating-point value as a string.

4.107 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 110.

Table 110 – Builtin fvar_value

Parameter name	Type	Direction	Usage	Description
source_var	reference	I	m	The name of the VARIABLE reference.
<return>	double	O	m	The value of the variable.

4.108 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 111.

Table 111 – Builtin get_acknowledgement

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	A message to be displayed.
ids	array of unsigned long	I	m	The item IDs of the VARIABLES used in the prompt message.
indices	array of unsigned long	I	m	The member IDs or array 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	The number of array elements in the arrays for ID s and indices. Set to zero if the arrays for ID s and indices are NULL.
<return>	long	O	m	One of the return codes specified in Table 366.

4.109 Builtin get_acknowledgement2**Purpose**

The Builtin get_acknowledgement2 displays the provided prompt message on the application display device and returns to the method after the user acknowledges the message. Variable values may be embedded in the message string with formatting information.

Lexical structure

```
get_acknowledgement2(prompt, blk_ids, blk_nums, ids, indices, id_count)
```

The lexical elements of the Builtin get_acknowledgement2 are shown in Table 112.

Table 112 – Builtin get_acknowledgement2

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	A message to be displayed.
blk_ids	array of unsigned long	I	m	The item IDs of the BLOCK_A instances used in the prompt message.
blk_nums	array of unsigned long	I	m	The occurrence numbers of the BLOCK_A instances used in the prompt message.
ids	array of unsigned long	I	m	The item IDs of the VARIABLES used in the prompt message.
indices	array of unsigned long	I	m	The member IDs or array 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	The number of array elements in the arrays for IDs and indices. Set to zero if the arrays for IDs and indices are NULL.
<return>	long	O	m	One of the return codes specified in Table 366.

4.110 Builtin get_block_instance_by_object_index**Purpose**

The Builtin get_block_instance_by_object_index returns the occurrence number of the BLOCK_A instance at the specified object index.

Lexical structure

```
get_block_instance_by_object_index(obj_idx, blk_num)
```

The lexical elements of the Builtin get_block_instance_by_object_index are shown in Table 113.

Table 113 – Builtin get_block_instance_by_object_index

Parameter name	Type	Direction	Usage	Description
obj_idx	unsigned long	I	m	The object index of the BLOCK_A instance to be returned.
blk_num	unsigned long	O	m	The occurrence number of the specified BLOCK_A instance.
<return>	long	O	m	One of the return codes specified in Table 366.

4.111 Builtin get_block_instance_by_tag**Purpose**

The Builtin get_block_instance_by_tag returns the occurrence number of the BLOCK_A instance with the specified tag. If the blockId parameter is zero the occurrence number of the block whose Block Tag matches the blockTag parameter will be returned. Otherwise, the occurrence number of the block whose DD Item ID matches the blockId parameter and whose Block Tag matches the blockTag parameter will be returned. If multiple blocks meet the specified criteria, the occurrence number of the block with the smallest object index will be returned.

Lexical structure

```
get_block_instance_by_tag(blockId, blockTag, blk_num)
```

The lexical elements of the Builtin `get_block_instance_by_tag` are shown in Table 114.

Table 114 – Builtin `get_block_instance_by_tag`

Parameter name	Type	Direction	Usage	Description
<code>blockId</code>	unsigned long	I	m	The item ID of the BLOCK_A instance to be returned.
<code>blockTag</code>	char[]	I	m	The tag of the BLOCK_A instance to be returned.
<code>blk_num</code>	unsigned long	O	m	The occurrence number of the specified BLOCK_A instance.
<return>	long	O	m	One of the return codes specified in Table 366.

4.112 Builtin `get_block_instance_count`

Purpose

The Builtin `get_block_instance_count` returns the number of BLOCK_A instances that are instantiated in the device. If the `blockId` parameter is zero, the total number of blocks currently instantiated in the device will be returned. Otherwise, the number of blocks whose DD Item ID matches the `blockId` parameter will be returned.

Lexical structure

`get_block_instance_count(blockId, count)`

The lexical elements of the Builtin `get_block_instance_count` are shown in Table 115.

Table 115 – Builtin `get_block_instance_count`

Parameter name	Type	Direction	Usage	Description
<code>blockId</code>	unsigned long	I	m	The item ID of the BLOCK_A instances to be counted.
<code>count</code>	unsigned long	O	m	The number of BLOCK_A instances of the specified type that are instantiated in the device.
<return>	long	O	m	One of the return codes specified in Table 366.

4.113 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 116.

Table 116 – Builtin `get_comm_error`

Parameter name	Type	Direction	Usage	Description
<return>	unsigned long	O	m	A communication error code.

4.114 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 117.

Table 117 – Builtin `get_comm_error_string`

Parameter name	Type	Direction	Usage	Description
error	unsigned long	I	m	The communication error code.
string	char[]	O	m	A description of the communication error.
maxlen	long	I	m	The maximal length of string.
<return>	long	O	m	One of the return codes specified in Table 366.

4.115 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 118.

Table 118 – Builtin `get_date`

Parameter name	Type	Direction	Usage	Description
data	char[]	O	m	The returned value of the VARIABLE. The structure of this octet buffer is defined in this specification. This output parameter shall point to valid storage.
size	long	O	m	The returned length of the VARIABLE. This output parameter shall point to valid storage.
<return>	long	O	m	One of the return codes specified in Table 366.

4.116 Builtin `get_date_lelem`**Purpose**

The Builtin `get_date_lelem` extracts a date value from a LIST.

Lexical structure

```
get_date_lelem(list_id, index, element_id, subelement_id, data, size)
```

The lexical elements of the Builtin `get_date_lelem` are shown in Table 119.

Table 119 – Builtin `get_date_lelem`

Parameter name	Type	Direction	Usage	Description
list_id	unsigned long		m	The item ID of the list.
index	int	I	m	The index of the list containing the item to be extracted.
element_id	unsigned long	I	m	The item ID of the element to be extracted.
subelement_id	unsigned long	I	m	The member ID of the element to be extracted.
data	char *	O	m	The buffer in which the data is to be returned.
size	int *	O	m	The size of the data returned.
<return>	long	O	m	One of the return codes specified in Table 366.

4.117 Builtin `get_date_lelem2`**Purpose**

The Builtin `get_date_lelem2` extracts a date value from an embedded list.

Lexical structure

```
get_date_lelem2(list_id, index, embedded_list_id, embedded_list_index,
element_id, subelement_id, data, size)
```

The lexical elements of the Builtin `get_date_lelem2` are shown in Table 120.

Table 120 – Builtin get_date_lelem2

Parameter name	Type	Direction	Usage	Description
list_id	unsigned long	I	m	The item ID of the list.
index	int	I	m	The index of the list containing the item to be extracted.
embedded_list_id	unsigned long	I	m	The item ID of the embedded list.
embedded_list_index	int	I	m	The index of the embedded list containing the item to be extracted.
element_id	unsigned long	I	m	The item ID of the element to be extracted.
subelement_id	unsigned long	I	m	The member ID of the element to be extracted.
data	char *	O	m	The buffer in which the data is to be returned.
size	int *	O	m	The size of the data returned.
<return>	long	O	m	One of the return codes specified in Table 366.

4.118 Builtin get_date_value**Purpose**

The Builtin get_date_value retrieves the current scaled value of the specified VARIABLE from the cache or device and returns it.

The value is an octet buffer that may contain up to eight characters and a VARIABLE of device type date-and-time, time, or duration.

Lexical structure

get_date_value(id, member_id, data, size)

The lexical elements of the Builtin get_date_value are shown in Table 121.

Table 121 – Builtin get_date_value

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	m	An ID of a VARIABLE reference.
member_id	unsigned long	I	m	A member ID of a BLOCK_A, PARAMETER_LIST, RECORD, VALUE_ARRAY, COLLECTION or REFERENCE_ARRAY.
data	char[]	O	m	A buffer of max. 8 octets for the result of the Builtin.
size	long	O	m	The size of the used buffer.
<return>	long	O	m	One of the return codes specified in Table 366.

4.119 Builtin get_date_value2**Purpose**

The Builtin get_date_value2 retrieves the current scaled value of the specified VARIABLE from the cache or device and returns it.

The value is an octet buffer that may contain up to eight characters and contain a VARIABLE of device type date-and-time, time, or duration.

Lexical structure

```
get_date_value2(blk_id, blk_num, id, member_id, data, size)
```

The lexical elements of the Builtin `get_date_value2` are shown in Table 122.

Table 122 – Builtin `get_date_value2`

Parameter name	Type	Direction	Usage	Description
<code>blk_id</code>	unsigned long	I	m	The item ID of the BLOCK_A instance to access.
<code>blk_num</code>	unsigned long	I	m	The occurrence number of the BLOCK_A instance to access.
<code>id</code>	unsigned long	I	m	An ID of a VARIABLE reference.
<code>member_id</code>	unsigned long	I	m	A member ID of a BLOCK_A, PARAMETER_LIST, RECORD, VALUE_ARRAY, COLLECTION or REFERENCE_ARRAY.
<code>data</code>	char[]	O	m	A buffer of max. 8 octets for the result of the Builtin.
<code>size</code>	long	O	m	The size of the used buffer.
<return>	long	O	m	One of the return codes specified in Table 366.

4.120 Builtin `GET_DD_REVISION`

Purpose

The Builtin `GET_DD_REVISION` gets the the DD_REVISION from the currently used device description as stated in the identification of the device description with the keyword DD_REVISION.

Lexical structure

```
GET_DD_REVISION (VOID)
```

The lexical element of the Builtin `GET_DD_REVISION` is shown in Table 123.

Table 123 – Builtin `GET_DD_REVISION`

Parameter name	Type	Direction	Usage	Description
<return>	int	o	m	The current DD_REVISION.

4.121 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. A textual representation of the error message is put into a provided buffer with a given maximum length.

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 124.

Table 124 – Builtin `get_dds_error`

Parameter name	Type	Direction	Usage	Description
string	char[]	O	m	An output buffer for the description of the error.
maxlen	long	I	m	The maximum length of the output buffer (string) in octets.
<return>	long	O	m	An error code generated by an earlier Builtin call.

4.122 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 125.

Table 125 – Builtin `GET_DEV_VAR_VALUE`

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	A message to be displayed.
device_var_name	reference	I	m	The referenced name of the VARIABLE, which can be edited.
<return>	int	O	m	One of the return codes specified in Table 367.

4.123 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 126.

Table 126 – Builtin `get_dev_var_value`

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	A message to be displayed.
global_var_ids	array of int	I	m	An array which contains unique identifiers of VARIABLE instance.
device_var_name	reference	I	m	The name of the VARIABLE, which can be edited.
<return>	int	O	m	One of the return codes specified in Table 367.

4.124 Builtin GET_DEVICE_REVISION**Purpose**

The Builtin `GET_DEVICE_REVISION` gets the `DEVICE_REVISION` from the currently used device description as stated in the identification of the device description with the keyword `DEVICE_REVISION`.

Lexical structure

```
GET_DEVICE_REVISION (VOID)
```

The lexical element of the Builtin `GET_DEVICE_REVISION` is shown in Table 127.

Table 127 – Builtin `GET_DEVICE_REVISION`

Parameter name	Type	Direction	Usage	Description
<return>	int	O	m	The current <code>DEVICE_REVISION</code> .

4.125 Builtin GET_DEVICE_TYPE**Purpose**

The Builtin `GET_DEVICE_TYPE` gets the numerical equivalent of the `DEVICE_TYPE` from the currently used device description as stated in the identification of the device description with the keyword `DEVICE_TYPE`.

Lexical structure

```
GET_DEVICE_TYPE (VOID)
```

The lexical element of the Builtin `GET_DEVICE_TYPE` is shown in Table 128.

Table 128 – Builtin `GET_DEVICE_TYPE`

Parameter name	Type	Direction	Usage	Description
<return>	int	O	m	The current <code>DEVICE_TYPE</code> .

4.126 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 129.

Table 129 – Builtin get_dictionary_string

Parameter name	Type	Direction	Usage	Description
dict_string_name	reference	I	m	The name of a dictionary entry.
string	char[]	I	m	A buffer for the result string.
max_str_len	int	I	m	The maximum length of the buffer.
<return>	int	O	m	One of the return codes specified in Table 367.

4.127 Builtin get_double

Purpose

The Builtin get_double retrieves the current unscaled double value of the VARIABLE being operated on.

It 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 130.

Table 130 – Builtin get_double

Parameter name	Type	Direction	Usage	Description
value	double	O	m	The returned value of the VARIABLE.
<return>	long	O	m	One of the return codes specified in Table 366.

4.128 Builtin get_double_ilem

Purpose

The Builtin get_double_ilem extracts a double value from a LIST.

Lexical structure

`get_double_lelem(list_id, index, element_id, subelement_id)`

The lexical element of the Builtin `get_double_lelem` is shown in Table 131.

Table 131 – Builtin `get_double_lelem`

Parameter name	Type	Direction	Usage	Description
list_id	unsigned long	I	m	The item ID of the list.
index	int	I	m	The index of the list containing the item to be extracted.
element_id	unsigned long	I	m	The item ID of the element to be extracted.
subelement_id	unsigned long	I	m	The member ID of the element to be extracted.
<return>	double	O	m	The specified value.

4.129 Builtin `get_double_lelem2`

Purpose

The Builtin `get_double_lelem2` extracts a double value from an embedded list.

Lexical structure

`get_double_lelem2(list_id, index, embedded_list_id, embedded_list_index, element_id, subelement_id)`

The lexical elements of the Builtin `get_double_lelem2` are shown in Table 132.

Table 132 – Builtin `get_double_lelem2`

Parameter name	Type	Direction	Usage	Description
list_id	unsigned long	I	m	The item ID of the list.
index	int	I	m	The index of the list containing the item to be extracted.
embedded_list_id	unsigned long	I	m	The item ID of the embedded list.
embedded_list_index	int	I	m	The index of the embedded list containing the item to be extracted.
element_id	unsigned long	I	m	The item ID of the element to be extracted.
subelement_id	unsigned long	I	m	The member ID of the element to be extracted.
<return>	double	O	m	The specified value.

4.130 Builtin `get_double_value`

Purpose

The Builtin `get_double_value` retrieves the current scaled value of the specified VARIABLE from the cache 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 cache or device.

Lexical structure

`get_double_value(id, member_id, value)`

The lexical elements of the Builtin `get_double_value` are shown in Table 133.

Table 133 – Builtin `get_double_value`

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	m	The item ID of the VARIABLE to access.
member_id	unsigned long	I	m	The member ID of the VARIABLE to access.
value	double	O	m	The returned value of the VARIABLE.
<return>	long	O	m	One of the return codes specified in Table 366.

4.131 Builtin `get_double_value2`

Purpose

The Builtin `get_double_value2` retrieves the current scaled value of the specified VARIABLE from the cache 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 cache or device.

Lexical structure

`get_double_value2(blk_id, blk_num, id, member_id, value)`

The lexical elements of the Builtin `get_double_value2` are shown in Table 134.

Table 134 – Builtin `get_double_value2`

Parameter name	Type	Direction	Usage	Description
blk_id	unsigned long	I	m	The item ID of the BLOCK_A instance to access.
blk_num	unsigned long	I	m	The occurrence number of the BLOCK_A instance to access.
id	unsigned long	I	m	The Item ID of the VARIABLE to access.
member_id	unsigned long	I	m	The member ID of the VARIABLE to access.
value	double	O	m	The returned value of the VARIABLE.
<return>	long	O	m	One of the return codes specified in Table 366.

4.132 Builtin get_enum_string

Purpose

The Builtin get_enum_string returns the string associated with the value of an enumerated or bit enumerated variable.

Lexical structure

get_enum_string(variable, value, string)

The lexical elements of the Builtin get_enum_string are shown in Table 135.

Table 135 – Builtin get_enum_string

Parameter name	Type	Direction	Usage	Description
variable	DD_ITEM	I	m	An enumerated or bit enumerated variable.
value	int	I	m	A value of the variable.
string	DD_STRING	O	m	String associated with the value of the variable.
<return>	int	O	m	One of the return codes specified in Table 366.

4.133 Builtin get_float

Purpose

The Builtin get_float retrieves the current unscaled floating-point value of the VARIABLE being operated on.

Gets the current value of an un-scaled floating-point VARIABLE as if it was directly read from a device.

The Builtin get_float can be called only by methods associated with a pre-edit, post-edit, post-read, or pre-write action. The methods associated with pre-edit, post-edit, post-read, or pre-write actions are run when a VARIABLE, which has one of these actions defined as an attribute, is read by an application, a user method, or an edit method.

Lexical structure

get_float(value)

The lexical elements of the Builtin get_float are shown in Table 136.

Table 136 – Builtin get_float

Parameter name	Type	Direction	Usage	Description
value	float	O	m	The returned value of the VARIABLE.
<return>	long	O	m	One of the return codes specified in Table 366.

4.134 Builtin get_float_lelem

Purpose

The Builtin get_float_lelem extracts a float value from a LIST.

Lexical structure

get_float_lelem(list_id, index, element_id, subelement_id)

The lexical elements of the Builtin get_float_lelem are shown in Table 137.

Table 137 – Builtin get_float_lelem

Parameter name	Type	Direction	Usage	Description
list_id	unsigned long	I	m	The item ID of the list.
index	int	I	m	The index of the list containing the item to be extracted.
element_id	unsigned long	I	m	The item ID of the element to be extracted.
subelement_id	unsigned long	I	m	The member ID of the element to be extracted.
<return>	float	O	m	The specified value.

4.135 Builtin get_float_lelem2

Purpose

The Builtin get_float_lelem2 extracts a float value from an embedded list.

Lexical structure

```
get_float_lelem2(list_id, index, embedded_list_id, embedded_list_index,
element_id, subelement_id)
```

The lexical elements of the Builtin get_float_lelem2 are shown in Table 138.

Table 138 – Builtin get_float_lelem2

Parameter name	Type	Direction	Usage	Description
list_id	unsigned long	I	m	The item ID of the list.
index	int	I	m	The index of the list containing the item to be extracted.
embedded_list_id	unsigned long	I	m	The item ID of the embedded list.
embedded_list_index	int	I	m	The index of the embedded list containing the item to be extracted.
element_id	unsigned long	I	m	The item ID of the element to be extracted.
subelement_id	unsigned long	I	m	The member ID of the element to be extracted.
<return>	float	O	m	The specified value.

4.136 Builtin get_float_value

Purpose

The Builtin get_float_value retrieves the current scaled value of the specified VARIABLE from the cache 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 139.

Table 139 – Builtin get_float_value

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	m	The item ID of the VARIABLE to access.
member_id	unsigned long	I	m	The member ID of the VARIABLE to access.
value	float	O	m	The returned value of the VARIABLE.
<return>	long	O	m	One of the return codes specified in Table 366.

4.137 Builtin get_float_value2**Purpose**

The Builtin get_float_value2 retrieves the current scaled value of the specified VARIABLE from the cache or device and returns it.

The parameters id and member_id are specified using the Builtins ITEM_ID and MEMBER_ID and the name or reference of the VARIABLE.

Lexical structure

get_float_value2(id, member_id, value)

The lexical elements of the Builtin get_float_value2 are shown in Table 140.

Table 140 – Builtin get_float_value2

Parameter name	Type	Direction	Usage	Description
blk_id	unsigned long	I	m	The item ID of the BLOCK_A instance to access.
blk_num	unsigned long	I	m	The occurrence number of the BLOCK_A instance to access.
id	unsigned long	I	m	The item ID of the VARIABLE to access.
member_id	unsigned long	I	m	The member ID of the VARIABLE to access.
value	float	O	m	The returned value of the VARIABLE.
<return>	long	O	m	One of the return codes specified in Table 366.

4.138 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 141.

Table 141 – Builtin GET_LOCAL_VAR_VALUE

Parameter name	Type	Direction	Usage	Description
prompt	char[]	l	m	A message to be displayed.
local_var_name	reference d	l	m	The name of the local VARIABLE to edit.
<return>	int	O	m	One of the return codes specified in Table 367.

4.139 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 142.

Table 142 – Builtin get_local_var_value

Parameter name	Type	Direction	Usage	Description
prompt	char[]	l	m	A message to be displayed.
global_var_ids	array of int	l	o	An array which contains unique identifiers of VARIABLE instance.
local_var_name	reference d	l	o	The name of the local VARIABLE to edit.
<return>	int	O	m	One of the return codes specified in Table 367.

4.140 Builtin GET_MANUFACTURER

Purpose

The Builtin GET_MANUFACTURER gets the numerical equivalent of the MANUFACTURER from the currently used device description as stated in the identification of the device description with the keyword MANUFACTURER.

Lexical Structure

GET_MANUFACTURER (VOID)

The lexical element of the Builtin GET_MANUFACTURER is shown in Table 143.

Table 143 – Builtin GET_MANUFACTURER

Parameter name	Type	Direction	Usage	Description
<return>	int	o	m	The current MANUFACTURER.

4.141 Builtin get_more_status

Purpose

The Builtin `get_more_status` will issue a command 48 to the device and return the response code, communications status, and command status octets in the status array provided. It will also return any data octets returned in the `more_data_info` array. This function will process the status/data octets in the same manner as the send command.

The calling function is responsible for allocating the arrays for the data to be returned. The parameter `more_data_status` is a 3-octet array, and the `more_data_info` is an array whose size is equal to the number of data octets that can be returned by the device in command 48 (maximum 25).

Lexical structure

`get_more_status(more_data_status, more_data_info)`

The lexical elements of the Builtin `get_more_status` are shown in Table 144.

Table 144 – Builtin get_more_status

Parameter name	Type	Direction	Usage	Description
<code>more_data_status</code>	<code>char[]</code>	I	m	Specifies if more info is available.
<code>more_data_info</code>	<code>char[]</code>	I	m	Specifies if more info is available.
<return>	<code>int</code>	O	m	One of the return codes specified in Table 367.

4.142 Builtin get_resolve_status

Purpose

The Builtin `get_resolve_status` returns the error number of the most recent “get_resolve_” Builtins. 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 Builtin is used for debugging purposes when creating methods.

It returns the codes for errors generated by Builtin `resolve_array_ref`, `resolve_block_ref`, `resolve_param_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 145.

Table 145 – Builtin get_resolve_status

Parameter name	Type	Direction	Usage	Description
<return>	unsigned long	O	m	One of the return codes specified in Table 366.

4.143 Builtin get_response_code

Purpose

The Builtin get_response_code returns the response code received from a device in the event of a Builtin communications call returning the error Builtin fail_on_response_code. It also returns the value reference (err_id, err_member_id) information associated with the original request. The original request is the last Builtin call that caused communication.

A response code is a value representing an application-specific error condition.

NOTE A response code value of zero is not considered an error response code. Each response code and response code type is defined in the EDD.

Gets the response code returned on the last communications request, and the item ID and member ID for the value being accessed when the response code was returned.

Lexical structure

get_response_code(resp_code, err_id, err_member_id)

The lexical elements of the Builtin get_response_code are shown in Table 146.

Table 146 – Builtin get_response_code

Parameter name	Type	Direction	Usage	Description
resp_code	unsigned long	O	m	The number of the returned response code.
err_id	unsigned long	O	m	The returned item ID of the item generating the response code.
err_member_id	unsigned long	O	m	The returned member ID of the item causing a response code.
<return>	long	O	m	One of the return codes specified in Table 366.

4.144 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 a 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 147.

Table 147 – Builtin get_response_code_string

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	m	The item ID of the item to find.
member_id	unsigned long	I	m	The member ID of the item to find.
code	unsigned long	I	m	The response code for which the descriptive string is to be found.
string	char*	O	m	The buffer for the returned response code description.
maxlen	long	I	m	The maximum length of the buffer.
<return>	long	O	m	One of the return codes specified in Table 366.

4.145 Builtin get_rspcode_string**Purpose**

The Builtin get_rspcode_string will return the response code string for the specified command and response code. If the string is longer than the maximum length defined in response_string_length, the string is truncated. The response code specified shall be valid for the indicated command.

Lexical structure

```
get_rspcode_string(cmd_number, code, string, length)
```

The lexical elements of the Builtin get_rspcode_string are shown in Table 148.

Table 148 – Builtin get_rspcode_string

Parameter name	Type	Direction	Usage	Description
cmd_number	int	I	m	The command number.
code	int	I	m	The response code.
string	DD_STRING	O	m	The buffer for the response code string.
length	int	I		The maximum length of the string buffer.
<return>	int	O	m	One of the return codes specified in Table 367.

4.146 Builtin get_rspcode_string_by_id**Purpose**

The Builtin get_rspcode_string_by_id returns a string representation of a given response_code (returned by ReadCommand, WriteCommand).

Lexical structure

```
get_rspcode_string_by_id(command, response_code, result)
```

The lexical elements of the Builtin get_rspcode_string_by_id are shown in Table 149.

Table 149 – Builtin get_rspcode_string_by_id

Parameter name	Type	Direction	Usage	Description
command	reference	I	m	The command from which the response_code was delivered.
response_code	int	I	m	The response_code value (as returned from ReadCommand or WriteCommand).
result	DD_STRING	O	m	String representation of the given response_code.
<return>	int	O	m	One of the return codes specified in Table 367.

4.147 Builtin get_signed**Purpose**

The Builtin get_signed retrieves the current unscaled signed value of the VARIABLE being operated on.

Gets the current value of a signed VARIABLE as if it was directly read from a device.

The Builtin get_signed can be called only by methods associated with a pre-edit, post-edit, post-read, or pre-write action. The methods associated with pre-edit, post-edit, post-read, or pre-write actions are run when a VARIABLE, which has one of these actions defined as an attribute, is read by an application, a user method, or an edit method.

Lexical structure

get_signed(value)

The lexical elements of the Builtin get_signed are shown in Table 150.

Table 150 – Builtin get_signed

Parameter name	Type	Direction	Usage	Description
value	long	I	m	The returned value of the signed VARIABLE.
<return>	long	O	m	One of the return codes specified in Table 366.

4.148 Builtin get_signed_lelem**Purpose**

The Builtin get_signed_lelem extracts a signed value from a LIST.

Lexical structure

get_signed_lelem(list_id, index, element_id, subelement_id)

The lexical elements of the Builtin get_signed_lelem are shown in Table 151.

Table 151 – Builtin get_signed_lelem

Parameter name	Type	Direction	Usage	Description
list_id	unsigned long	I	m	The item ID of the list.
index	int	I	m	The index of the list containing the item to be extracted.
element_id	unsigned long	I	m	The item ID of the element to be extracted.
subelement_id	unsigned long	I	m	The member ID of the element to be extracted.
<return>	long	O	m	The specified value.

4.149 Builtin get_signed_lelem2**Purpose**

The Builtin get_signed_lelem2 extracts a signed value from an embedded list.

Lexical structure

```
get_signed_lelem2(list_id, index, embedded_list_id, embedded_list_index,
element_id, subelement_id)
```

The lexical elements of the Builtin get_signed_lelem2 are shown in Table 152.

Table 152 – Builtin get_signed_lelem2

Parameter name	Type	Direction	Usage	Description
list_id	unsigned long	I	m	The item ID of the list.
index	int	I	m	The index of the list containing the item to be extracted.
embedded_list_id	unsigned long	I	m	The item ID of the embedded list.
embedded_list_index	int	I	m	The index of the embedded list containing the item to be extracted.
element_id	unsigned long	I	m	The item ID of the element to be extracted.
subelement_id	unsigned long	I	m	The member ID of the element to be extracted.
<return>	long	O	m	The specified value.

4.150 Builtin get_signed_value**Purpose**

The Builtin get_signed_value retrieves the current scaled value of the specified VARIABLE from the cache or device and returns it.

The 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 cache or device.

Lexical structure

```
get_signed_value(id, member_id, value)
```

The lexical elements of the Builtin `get_signed_value` are shown in Table 153.

Table 153 – Builtin `get_signed_value`

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	m	The item ID of the VARIABLE to access.
member_id	unsigned long	I	m	The member ID of the VARIABLE to access.
value	long	O	m	The returned value of the signed VARIABLE.
<return>	long	O	m	One of the return codes specified in Table 366.

4.151 Builtin `get_signed_value2`

Purpose

The Builtin `get_signed_value2` retrieves the current scaled value of the specified VARIABLE from the cache or device and returns it.

The 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 cache or device.

Lexical structure

`get_signed_value2(blk_id, blk_num, id, member_id, value)`

The lexical elements of the Builtin `get_signed_value2` are shown in Table 154.

Table 154 – Builtin `get_signed_value2`

Parameter name	Type	Direction	Usage	Description
blk_id	unsigned long	I	m	The item ID of the BLOCK_A instance to access.
blk_num	unsigned long	I	m	The occurrence number of the BLOCK_A instance to access.
id	unsigned long	I	m	The item ID of the VARIABLE to access.
member_id	unsigned long	I	m	The member ID of the VARIABLE to access.
value	long	O	m	The returned value of the signed VARIABLE.
<return>	long	O	m	One of the return codes specified in Table 366.

4.152 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 155.

Table 155 – Builtin get_status_code_string

Parameter name	Type	Direction	Usage	Description
variable_name	char[]	I	m	The name of the VARIABLE to access.
status_code	int	I	m	The value of interesting status code.
status_string	char *	O	m	The buffer of the returned value of the signed VARIABLE.
status_string_length	int	I	m	The maximum size of the buffer.
<return>	VOID	—	—	—

4.153 Builtin get_status_string**Purpose**

The Builtin get_status_string searches the members of the referenced ENUMERATED or BIT_ENUMERATED VARIABLE and returns the description of the member matching the status in the given buffer.

The Builtin gets the string associated with an ENUMERATED or BIT_ENUMERATED value. The most common use of ENUMERATED and BIT_ENUMERATED VARIABLES is to get a string associated with a status bit defined in the device for a particular status. The strings associated with ENUMERATED and BIT_ENUMERATED VARIABLES are defined in the EDDL.

The parameters id and member_id are specified using the Builtins ITEM_ID and MEMBER_ID and the name or reference of the VARIABLE in question.

Gets the description of a specific status in a specific device.

Lexical structure

```
get_status_string(id, member_id, status, string, maxlen)
```

The lexical elements of the Builtin get_status_string are shown in Table 156.

Table 156 – Builtin get_status_string

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	m	The item ID of the VARIABLE to access.
member_id	unsigned long	I	m	The member ID of the VARIABLE to access.
status	unsigned long	I	m	The value to search.
string	char*	O	m	The buffer for the returned string.
maxlen	long	I	m	The maximum size of the buffer.
<return>	long	O	m	One of the return codes specified in Table 366.

4.154 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. The Builtin strings are defined in the standard .dct file.

Lexical structure

`get_stdidict_string(id, string, maxlen)`

The lexical elements of the Builtin `get_stdidict_string` are shown in Table 157.

Table 157 – Builtin `get_stdidict_string`

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	m	The text dictionary identifier.
string	char*	O	m	The returned buffer for the string.
maxlen	long	I	m	The maximum size of the buffer.
<return>	long	O	m	One of the return codes specified in Table 366.

4.155 Builtin `get_string`

Purpose

The Builtin `get_string` retrieves the current, unscaled, string value of the VARIABLE being operated on.

Gets the current value of the string VARIABLE as if it was directly read from a device. The VARIABLE shall be a valid parameter. The VARIABLE shall have a data type of char and be an octet buffer which can contain various non-ANSI data types such as:

- ASCII
- PASSWORD
- EUC (extended unicode)
- BITSTRING

Device VARIABLES of these data types shall be handled by the method because they are not supported by the programming language-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 158.

Table 158 – Builtin `get_string`

Parameter name	Type	Direction	Usage	Description
string	char*	O	m	The buffer for the returned string.
len	long*	I/O	m	The maximum size of the string. The returned length of the string.
<return>	int	O	m	One of the return codes specified in Table 366.

4.156 Builtin get_string_lelem

Purpose

The Builtin get_string_lelem extracts a string from a LIST.

Lexical structure

```
get_string_lelem(list_id, index, element_id, subelement_id, string, length)
```

The lexical elements of the Builtin get_string_lelem are shown in Table 159.

Table 159 – Builtin get_string_lelem

Parameter name	Type	Direction	Usage	Description
list_id	unsigned long	I	m	The item ID of the list.
index	int	I	m	The index of the list containing the item to be extracted.
element_id	unsigned long	I	m	The item ID of the element to be extracted.
subelement_id	unsigned long	I	m	The member ID of the element to be extracted.
string	char *	O	m	The buffer in which the data is to be returned.
length	int *	O	m	The size of the data returned.
<return>	long	O	m	The specified value.

4.157 Builtin get_string_lelem2

Purpose

The Builtin get_string_lelem2 extracts a string from an embedded list.

Lexical structure

```
get_string_lelem2(list_id, index, embedded_list_id, embedded_list_index,
element_id, subelement_id, string, length)
```

The lexical elements of the Builtin get_string_lelem2 are shown in Table 160.

Table 160 – Builtin get_string_lelem2

Parameter name	Type	Direction	Usage	Description
list_id	unsigned long	I	m	The item ID of the list.
index	int	I	m	The index of the list containing the item to be extracted.
embedded_list_id	unsigned long	I	m	The item ID of the embedded list.
embedded_list_index	int	I	m	The index of the embedded list containing the item to be extracted.
element_id	unsigned long	I	m	The item ID of the element to be extracted.
subelement_id	unsigned long	I	m	The member ID of the element to be extracted.
string	char*	O	m	The buffer in which the data is to be returned.
length	int*	O	m	The size of the data returned.
<return>	long	O	m	The specified value.

4.158 Builtin get_string_value

Purpose

The Builtin get_string_value retrieves the current scaled value of the specified VARIABLE from the cache or device and returns it.

The 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 161.

Table 161 – Builtin get_string_value

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	m	The ID of a VARIABLE reference.
member_id	unsigned long	I	m	The member ID of a BLOCK_A, PARAMETER_LIST, RECORD, VALUE_ARRAY, COLLECTION or REFERENCE_ARRAY.
string	char*	O	m	The buffer of result string.
len	long	O	m	The size of the result string.
<return>	long	O	m	One of the return codes specified in Table 366.

4.159 Builtin get_string_value2

Purpose

The Builtin get_string_value2 retrieves the current scaled value of the specified VARIABLE from the Cache or device and returns it.

The parameters id and member_id are specified using the Builtins ITEM_ID and MEMBER_ID and the name or reference of the VARIABLE in question.

Lexical structure

```
get_string_value2(blk_id, blk_num, id, member_id, string, len)
```

The lexical elements of the Builtin `get_string_value2` are shown in Table 162.

Table 162 – Builtin `get_string_value2`

Parameter name	Type	Direction	Usage	Description
blk_id	unsigned long	I	m	The item ID of the BLOCK_A instance to access.
blk_num	unsigned long	I	m	The occurrence number of the BLOCK_A instance to access.
id	unsigned long	I	m	The ID of a VARIABLE reference.
member_id	unsigned long	I	m	The member ID of a BLOCK_A, PARAMETER_LIST, RECORD, VALUE_ARRAY, COLLECTION or REFERENCE_ARRAY.
string	char*	O	m	Specifies the buffer of result string.
len	long	O	m	The size of the result string.
<return>	long	O	m	One of the return codes specified in Table 366.

4.160 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 163.

Table 163 – Builtin `GET_TICK_COUNT`

Parameter name	Type	Direction	Usage	Description
<return>	long	O	m	The time in ms since the last system boot.

4.161 Builtin `get_transfer_status`**Purpose**

The Builtin `get_transfer_status` function is used to poll for completion of the EDD-Item currently being transferred.

Lexical structure

```
get_transfer_status(handle, direction, transfer_status)
```

The lexical elements of the Builtin `get_transfer_status` are shown in Table 164.

Table 164 – Builtin get_transfer_status

Parameter name	Type	Direction	Usage	Description
handle	unsigned int	I	m	The handle for the block transfer.
direction	unsigned char	I	m	Specifies the direction being polled: 0x01 for read 0x02 for write 0x03 for read and write
transfer_status	unsigned int[4]	O	m	The returned transfer status containing the following information: • response code • function code • master-byte-count, • device-byte-count
<return>	int	O	m	One of the return codes specified in Table 367.

4.162 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 165.

Table 165 – Builtin get_unsigned

Parameter name	Type	Direction	Usage	Description
value	unsigned long	O	m	The returned value.
<return>	long	O	m	One of the return codes specified in Table 366.

4.163 Builtin get_unsigned_lelem

Purpose

The Builtin get_unsigned_lelem extracts an unsigned value from a LIST.

Lexical structure

get_unsigned_lelem(list_id, index, element_id, subelement_id)

The lexical elements of the Builtin get_unsigned_lelem are shown in Table 166.

Table 166 – Builtin get_unsigned_lelem

Parameter name	Type	Direction	Usage	Description
list_id	unsigned long	I	m	The item ID of the list.
index	int	I	m	The index of the list containing the item to be extracted.
element_id	unsigned long	I	m	The item ID of the element to be extracted.
subelement_id	unsigned long	I	m	The member ID of the element to be extracted.
<return>	unsigned long	O	m	The specified value.

4.164 Builtin get_unsigned_lelem2**Purpose**

The Builtin get_unsigned_lelem2 extracts an unsigned value from an embedded list.

Lexical structure

```
get_unsigned_lelem2(list_id, index, embedded_list_id, embedded_list_index,
element_id, subelement_id)
```

The lexical elements of the Builtin get_unsigned_lelem2 are shown in Table 167.

Table 167 – Builtin get_unsigned_lelem2

Parameter name	Type	Direction	Usage	Description
list_id	unsigned long	I	m	The item ID of the list.
index	int	I	m	The index of the list containing the item to be extracted.
embedded_list_id	unsigned long	I	m	The item ID of the embedded list.
embedded_list_index	int	I	m	The index of the embedded list containing the item to be extracted.
element_id	unsigned long	I	m	The item ID of the element to be extracted.
subelement_id	unsigned long	I	m	The member ID of the element to be extracted.
<return>	unsigned long	O	m	Returns the specified value.

4.165 Builtin get_unsigned_value**Purpose**

The Builtin get_unsigned_value retrieves the current scaled value of the specified VARIABLE from the Cache 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 the programming language-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 168.

Table 168 – Builtin `get_unsigned_value`

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	m	The item ID of the variable to access.
member_id	unsigned long	I	m	The member ID of the variable to access.
value	unsigned long	O	m	The returned value of the variable.
<return>	long	O	m	One of the return codes specified in Table 366.

4.166 Builtin `get_unsigned_value2`

Purpose

The Builtin `get_unsigned_value2` retrieves the current scaled value of the specified VARIABLE from the Cache 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 the programming language-C. The variable shall be a valid parameter.

Lexical structure

`get_unsigned_value2(blk_id, blk_num, id, member_id, value)`

The lexical elements of the Builtin `get_unsigned_value2` are shown in Table 169.

Table 169 – Builtin get_unsigned_value2

Parameter name	Type	Direction	Usage	Description
blk_id	unsigned long	I	m	The item ID of the BLOCK_A instance to access.
blk_num	unsigned long	I	m	The occurrence number of the BLOCK_A instance to access.
id	unsigned long	I	m	The item ID of the variable to access.
member_id	unsigned long	I	m	The member ID of the variable to access.
value	unsigned long	O	m	The returned value of the variable.
<return>	long	O	m	One of the return codes specified in Table 366.

4.167 Builtin get_variable_string**Purpose**

The Builtin get_variable_string will get a textual representation of the referenced VARIABLE.

Lexical structure

```
get_variable_string(device_var_name, destination_string)
```

The lexical elements of the Builtin get_variable_string are shown in Table 170.

Table 170 – Builtin get_variable_string

Parameter name	Type	Direction	Usage	Description
device_var_name	reference	I	m	The name of the referenced VARIABLE.
destination_string	string	O	m	Buffer for the textual representation of the referenced VARIABLE.
<return>	int	O	m	One of the return codes specified in Table 367.

4.168 Builtin GetCurrentDate**Purpose**

The Builtin GetCurrentDate returns the EDD application's best approximation of the current calendar date.

Lexical structure

```
GetCurrentDate()
```

The lexical element of the Builtin GetCurrentDate is shown in Table 171.

Table 171 – Builtin GetCurrentDate

Parameter name	Type	Direction	Usage	Description
<return>	time_t	o	m	The current calendar date.

4.169 Builtin GetCurrentDateAndTime

Purpose

The Builtin GetCurrentDateAndTime returns the EDD application's best approximation of the current calendar date and time.

Lexical structure

```
GetCurrentDateAndTime()
```

The lexical element of the Builtin GetCurrentDateAndTime is shown in Table 172.

Table 172 – Builtin GetCurrentDateAndTime

Parameter name	Type	Direction	Usage	Description
<return>	time_t	o	m	The current calendar date and time.

4.170 Builtin GetCurrentTime

Purpose

The Builtin GetCurrentTime gets the current time from the host application's best approximation of the current calendar time.

Lexical structure

```
GetCurrentTime(VOID)
```

The lexical element of the Builtin GetCurrentTime is shown in Table 173.

Table 173 – Builtin GetCurrentTime

Parameter name	Type	Direction	Usage	Description
<return>	time_t	o	m	The current calendar time.

4.171 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 174.

Table 174 – Builtin iassign

Parameter name	Type	Direction	Usage	Description
device_var	reference	l	m	The name of the variable to access.
new_value	int	l	m	The new value of the VARIABLE.
<return>	int	O	m	One of the return codes specified in Table 367.

4.172 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 175.

Table 175 – Builtin igetval

Parameter name	Type	Direction	Usage	Description
<return>	int	O	m	The value of the variable.

4.173 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 176.

Table 176 – Builtin IGNORE_ALL_COMM_STATUS

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.174 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 177.

Table 177 – Builtin IGNORE_ALL_DEVICE_STATUS

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.175 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 178.

Table 178 – Builtin IGNORE_ALL_RESPONSE_CODES

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.176 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 179.

Table 179 – Builtin IGNORE_COMM_ERROR

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.177 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 180.

Table 180 – Builtin IGNORE_COMM_STATUS

Parameter name	Type	Direction	Usage	Description
comm_status	int	l	m	The new retry and abort mask for ignoring.
<return>	VOID	—	—	—

4.178 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 181.

Table 181 – Builtin IGNORE_DEVICE_STATUS

Parameter name	Type	Direction	Usage	Description
device_status	int	I	m	The new retry and abort mask for ignoring.
<return>	VOID	—	—	—

4.179 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 182.

Table 182 – Builtin IGNORE_NO_DEVICE

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.180 Builtin IGNORE_RESPONSE_CODE

Purpose

The Builtin IGNORE_RESPONSE_CODE will clear the correct bit(s) in the response code masks such that the specified response code value will be ignored. The response code is defined to be the first data octet returned in a transaction, when bit 7 of this octet is 0.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this function is used by the Builtins `send`, `send_trans`, `send_command`, `send_command_trans`, `ext_send_command`, `ext_send_command_trans`, `get_more_status` and `display`.

Lexical structure

`IGNORE_RESPONSE_CODE(response_code)`

The lexical elements of the Builtin `IGNORE_RESPONSE_CODE` are shown in Table 183.

Table 183 – Builtin `IGNORE_RESPONSE_CODE`

Parameter name	Type	Direction	Usage	Description
<code>response_code</code>	int	I	m	Specifies the new retry and abort mask for ignoring
<return>	VOID	—	—	—

4.181 Builtin `int_value`

Purpose

The Builtin `int_value` will return the value of the specified device variable. The variable shall be valid and of type integer.

Lexical structure

`int_value(variable)`

The lexical elements of the Builtin `int_value` are shown in Table 184.

Table 184 – Builtin `int_value`

Parameter name	Type	Direction	Usage	Description
<code>variable</code>	reference	I	m	The VARIABLE whose value shall be retrieved.
<return>	int	O	m	The value of the variable.

4.182 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 standard as an invalid double value.

Lexical structure

`is_NaN(dvalue)`

The lexical elements of the Builtin `is_NaN` are shown in Table 185.

Table 185 – Builtin `is_NaN`

Parameter name	Type	Direction	Usage	Description
<code>dvalue</code>	double	I	m	The name of the variable.
<return>	long	O	m	One if the <code>dvalue</code> is equal to NaN, zero otherwise.

4.183 Builtin `isetval`

Purpose

The Builtin `isetval` will set the value of the device variable of type `INTEGER`, `ENUMERATED`, `BIT_ENUMERATED`, `UNSIGNED_INTEGER` and `INDEX` 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 186.

Table 186 – Builtin `isetval`

Parameter name	Type	Direction	Usage	Description
value	int	I	m	The new value of the variable.
<return>	int	O	m	The value of the variable.

4.184 Builtin `isOffline`

Purpose

The builtin `isOffline` returns a non-zero value when the Host Application state (within the context of the calling method) is currently "Offline".

Lexical structure

`isOffline()`

The lexical element of the Builtin `isOffline` is shown in Table 187.

Table 187 – Builtin `isOffline`

Parameter name	Type	Direction	Usage	Description
<return>	int	O	m	1 if the EDD application cannot communicate with the device, 0 if the EDD application can communicate with the device.

4.185 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 188.

Table 188 – Builtin ITEM_ID

Parameter name	Type	Direction	Usage	Description
name	reference	I	m	The name of the variable.
<return>	unsigned long	O	m	The VARIABLE's item ID.

4.186 Builtin itoa (version A)**Purpose**

The Builtin converts an integer value into a text string.

Lexical structure

```
itoa(i)
```

The lexical elements of the Builtin itoa are shown in Table 189.

Table 189 – Builtin itoa (version A)

Parameter name	Type	Direction	Usage	Description
i	int	I	m	An integer value to convert to text.
<return>	char[]	O	m	The returned text

4.187 Builtin itoa (version B)**Purpose**

The Builtin itoa converts an integer value into a text string.

Lexical structure

```
itoa(i, destination, radix)
```

The lexical elements of the Builtin itoa are shown in Table 190.

Table 190 – Builtin itoa (version B)

Parameter name	Type	Direction	Usage	Description
i	int	I	m	An integer value to convert to text.
destination	char[]	I	m	The string in which text is to be stored.
radix	int	I	m	The base of the value (2 = binary, 8 = octal, 10 = decimal, 16 = hexadecimal).
<return>	char[]	O	m	The returned text.

4.188 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 191.

Table 191 – Builtin ivar_value

Parameter name	Type	Direction	Usage	Description
source_var_name	reference	I	m	The name of the variable.
<return>	int	O	m	The value of the variable.

4.189 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 192.

Table 192 – Builtin lassign

Parameter name	Type	Direction	Usage	Description
var_name	reference	I	m	The name of the VARIABLE.
new_value	int	I	m	The new value of the variable.
<return>	int	O	m	One of the return codes specified in Table 367.

4.190 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(VOID)
```

The lexical element of the Builtin lgetval is shown in Table 193.

Table 193 – Builtin lgetval

Parameter name	Type	Direction	Usage	Description
<return>	int	O	m	The value of the variable.

4.191 Builtin ListDeleteElementAt**Purpose**

The Builtin ListDeleteElementAt will delete an element from a list. If the list is a member of a FILE, the specified element will be removed from the persistent store.

If a COUNT is explicitly declared on the specified LIST, any variables referenced by the COUNT attribute will not be updated. The method is responsible for updating the COUNT when elements are deleted from the list. If a COUNT is not explicitly declared, the COUNT attribute will be automatically updated.

If the index parameter is less than zero or greater than the number of elements in the list, an error will be returned. If the list parameter does not refer to a LIST instance, an error will be returned.

Lexical structure

```
ListDeleteElementAt(list, index)
```

The lexical elements of the Builtin ListDeleteElementAt are shown in Table 194.

Table 194 – Builtin ListDeleteElementAt

Parameter name	Type	Direction	Usage	Description
list	reference	I	m	The name of a LIST instance.
index	int	I	m	The index of the element to be deleted.
<return>	long	O	m	One of the return codes specified in Table 367. EDD Applications may return also the code in Table 366 for diagnostic purposes.

4.192 Builtin ListDeleteElementAt2

Purpose

The Builtin ListDeleteElementAt2 deletes elements from an embedded list. If the list is a member of a FILE, the specified element will be removed from the persistent store.

Lexical structure

```
ListDeleteElementAt2(list_id, list_index, embedded_list_id, embedded_index, embedded_count)
```

The lexical elements of the Builtin ListDeleteElementAt2 are shown in Table 195.

Table 195 – Builtin ListDeleteElementAt2

Parameter name	Type	Direction	Usage	Description
list_id	unsigned long	I	m	The item of the list.
list_index	int	I	m	The index of the list containing the embedded list.
embedded_list_id	unsigned long	I	m	The item ID of the embedded_list.
embedded_index	int	I	m	The index of the first element to delete.
embedded_count	int	I	m	The number of elements to delete.
<return>	long	O	m	One of the return codes specified in Table 367. EDD Applications may return also the code in Table 366 for diagnostic purposes.

4.193 Builtin ListInsert

Purpose

The Builtin ListInsert will insert an element to a list. If the list is a member of a FILE, the specified element will be added to the persistent store.

If COUNT and/or CAPACITY attributes are explicitly declared on the specified LIST, any variables referenced by these attributes will not be updated. The method is responsible for updating the COUNT and CAPACITY when elements are inserted in the list. If COUNT and/or

CAPACITY attributes are not explicitly declared, these attributes will be updated automatically.

If the index parameter is less than zero or greater than the capacity of the list, an error will be returned. If the list parameter does not refer to a LIST instance, an error will be returned.

Lexical structure

ListInsert(list, index, item)

The lexical elements of the Builtin ListInsert are shown in IEC 61804-3:–, Table 150.

Table 196 – Builtin ListInsert

Parameter name	Type	Direction	Usage	Description
list	reference	I	m	The name of a LIST instance.
index	int	I	m	The index of the requested element.
item	reference	I	m	The name of the item to be inserted.
<return>	long	O	m	One of the return codes specified in Table 367. EDD Applications may return also the code in Table 366 for diagnostic purposes.

4.194 Builtin ListInsert2

Purpose

The Builtin ListInsert2 inserts an element into an embedded list. If the list is a member of a FILE, the specified element will be added to the persistent store.

Lexical structure

ListInsert2(list_id, list_index, embedded_list_id, embedded_index, item_id)

The lexical elements of the Builtin ListInsert2 are shown in Table 197.

Table 197 – Builtin ListInsert2

Parameter name	Type	Direction	Usage	Description
list_id	unsigned long	I	m	The item of the list.
list_index	int	I	m	The index of the list containing the embedded list.
embedded_list_id	unsigned long	I	m	The item ID of the embedded list.
embedded_index	int	I	m	The index where the item should be inserted.
item_id	unsigned long	I	m	The item ID of the item to be inserted.
<return>	long	O	m	One of the return codes specified in Table 367. EDD Applications may return also the code in Table 366 for diagnostic purposes.

4.195 Builtin log

Purpose

The Builtin log returns the natural logarithm of a floating-point value.

Lexical structure

$\log(x)$

The lexical elements of the Builtin log are shown in Table 198.

Table 198 – Builtin log

Parameter name	Type	Direction	Usage	Description
x	double	I	m	A floating-point value.
<return>	double	O	m	The natural logarithm.

4.196 Builtin LOG_MESSAGE

Purpose

The Builtin LOG_MESSAGE inserts a message into a logging protocol.

Lexical structure

LOG_MESSAGE(priority, message)

The lexical elements of the Builtin LOG_MESSAGE are shown in Table 199.

Table 199 – Builtin LOG_MESSAGE

Parameter name	Type	Direction	Usage	Description
priority	int	I	m	The priority of the message (0 = error, 1 = warning, 2 = low).
message	char[]	I	m	The message string to insert into the logging protocol.
<return>	VOID	—	—	—

4.197 Builtin log10

Purpose

The Builtin exp returns the base 10 logarithm of a floating-point value.

Lexical structure

$\log_{10}(x)$

The lexical elements of the Builtin log10 are shown in Table 200.

Table 200 – Builtin log10

Parameter name	Type	Direction	Usage	Description
x	double	I	m	A floating-point value.
<return>	double	O	m	The base 10 logarithm.

4.198 Builtin log2

Purpose

The Builtin log2 returns the base 2 logarithm of a floating-point value.

Lexical structure

$\log_2(x)$

The lexical elements of the Builtin $\log_2$ are shown in Table 201.

Table 201 – Builtin $\log_2$

Parameter name	Type	Direction	Usage	Description
x	double	I	m	A floating-point value.
<return>	double	O	m	The base 2 logarithm.

4.199 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 202.

Table 202 – Builtin long_value

Parameter name	Type	Direction	Usage	Description
source_var_name	reference	I	m	The name of the variable.
<return>	int	O	m	The value of the variable.

4.200 Builtin LongToByte

Purpose

The Builtin LongToByte will provide a vector of unsigned char values representing the particular elements of a long value.

Lexical structure

LongToByte(in, out1, out2, out3, out4)

The lexical element of the Builtin LongToByte is shown in Table 203.

Table 203 – Builtin LongToByte

Parameter name	Type	Direction	Usage	Description
in	long	I	m	The long value to convert.
out1	unsigned char	O	m	First element of the long value (most significant byte).
out2	unsigned char	O	m	Second element of the long value.
out3	unsigned char	O	m	Third element of the long value.
out4	unsigned char	O	m	Fourth element of the long value (least significant byte).
<return>	VOID	—	—	—

4.201 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 204.

Table 204 – Builtin lsetval

Parameter name	Type	Direction	Usage	Description
value	int	I	m	The new value of the VARIABLE.
<return>	int	O	m	The value of the variable.

4.202 Builtin lvar_value

Purpose

The Builtin lvar_value will return the value of the specified device variable. The variable shall be valid.

Lexical structure

lvar_value(source_var_name)

The lexical elements of the Builtin lvar_value are shown in Table 205.

Table 205 – Builtin lvar_value

Parameter name	Type	Direction	Usage	Description
source_var_name	reference	I	m	The name of the VARIABLE.
<return>	long	O	m	The value of the variable.

4.203 Builtin Make_Time

Purpose

The Builtin Make_Time creates a time_t from its input parameters.

Lexical structure

Make_Time(year, month, day, hour, minute, second, isDST)

The lexical elements of the Builtin Make_Time are shown in Table 206.

Table 206 – Builtin Make_Time

Parameter name	Type	Direction	Usage	Description
year	int	I	m	The year.
month	int	I	m	The month.
day	int	I	m	The day.
hour	int	I	m	The hour.
minute	int	I	m	The minute.
second	int	I	m	The second.
isDST	int	I	m	Specifies whether daylight savings time is in effect. If zero, daylight savings time is not in effect; otherwise, daylight savings time is in effect.
<return>	time_t	O	m	The time_t that was created or -1 if an invalid parameter is detected.

4.204 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, or COLLECTION.

Lexical structure

MEMBER_ID (name)

The lexical elements of the Builtin MEMBER_ID are shown in Table 207.

Table 207 – Builtin MEMBER_ID

Parameter name	Type	Direction	Usage	Description
name	reference	I	m	The name of an item member.
<return>	unsigned long	O	m	The referenced member ID.

4.205 Builtin MenuDisplay

Purpose

The Builtin MenuDisplay will display a MENU instance to the user and a supplied list of options for the user to select from. Once a selection is made, the option position in the option list is returned. The position in the list is zero-based. For example, if the provided list contains two items and the first item is selected, this function will return a 0. If the second item is selected, a 1 is returned. The options are passed to the function as a single string with semicolon separators between the options. The MENU shall be treated as a MENU with STYLE DIALOG regardless of the actual STYLE of the MENU.

Lexical structure

MenuDisplay(menu, options, selection)

If the EDD application is unable to display the menu, an error will be returned.

The lexical elements of the Builtin MenuDisplay are shown in Table 208.

Table 208 – Builtin MenuDisplay

Parameter name	Type	Direction	Usage	Description
menu	reference	I	m	The MENU to be displayed.
options	char[]	I	m	The option string, consisting of separated text, that defines a set of options presented to the user.
selection	long*	O	m	The zero-based selected option (see options parameter).
<return>	long	O	m	One of the return codes specified in Table 367. EDD Applications may return also the code in Table 366 for diagnostic purposes.

4.206 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 209.

Table 209 – Builtin method_abort

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	A message to be displayed.
<return>	VOID	—	—	—

4.207 Builtin nan**Purpose**

The Builtin nan returns a "not a number" value as a double with a fractional bit pattern based on a string. If the character sequence begins with a quiet or signaling NaN prefix letter, then any following hex digits in the character sequence shall be valid for that type of NaN.

The string can be:

- "" (an empty string) generates the default NaN 0x7FF4000000000000.
- "Q" or "q" equals the default quiet NaN.
- "S" or "s" equals the default signaling NaN.

- "l_xxxxx" equals where l is either a "Q", "q", "S", or "s" followed by an underscore and the most significant hex digits of the fraction. Trailing zeros may be omitted.

Lexical structure

nan(value)

The lexical elements of the Builtin nan are shown in Table 210.

Table 210 – Builtin nan

Parameter name	Type	Direction	Usage	Description
value	DD_STRING	I	m	The "not a number" value to be returned.
<return>	double	O	m	The specified value.

4.208 Builtin NaN_value

Purpose

The Builtin NaN_value returns the value of "not a number".

Lexical structure

NaN_value(value)

The lexical elements of the Builtin NaN_value are shown in Table 211.

Table 211 – Builtin NaN_value

Parameter name	Type	Direction	Usage	Description
value	double	O	m	A pointer through which the value is returned.
<return>	long	O	m	The specified value.

4.209 Builtin nanf

Purpose

The Builtin nanf returns a "not a number" value as a float with a fractional bit pattern based on a string. If the character sequence begins with a quiet or signaling NaN prefix letter, then any following hex digits in the character sequence shall be valid for that type of NaN.

The string can be:

- "" (an empty string), generates the default NaN 0x7FA00000.
- "Q" or "q" equals the default quiet NaN.
- "S" or "s" equals the default signaling NaN.
- "l_xxxxx" equals where l is either a "Q", "q", "S", or "s" followed by an underscore and the most significant hex digits of the fraction. Trailing zeros maybe omitted.

Lexical structure

nanf(value)

The lexical elements of the Builtin nanf are shown in Table 212.

Table 212 – Builtin nanf

Parameter name	Type	Direction	Usage	Description
value	DD_STRING	I	m	The "not a number" value to be returned.
<return>	float	O	m	The specified value.

4.210 Builtin ObjectReference**Purpose**

The Builtin ObjectReference returns a reference to a COMPONENT instance. The return value can be used to access data objects and methods of other COMPONENT instances.

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(navigation)

The lexical elements of the Builtin ObjectReference are shown in Table 213.

Table 213 – Builtin ObjectReference

Parameter name	Type	Direction	Usage	Description
navigation	One of the following values: SELF CHILD FIRST LAST PARENT NEXT PREV	I	m	See IEC 61804-3 for the direction of navigation.
<return>	OBJECT_REFERENCE	O	m	A reference to a COMPONENT instance.

4.211 Builtin openTransferPort**Purpose**

The Builtin openTransferPort opens the block transfer port.

Lexical structure

open_transfer_port(port, handle)

The lexical elements of the Builtin openTransferPort are shown in Table 214.

Table 214 – Builtin openTransferPort

Parameter name	Type	Direction	Usage	Description
port	unsigned char	I	m	The block transfer port number.
handle	unsigned int	O	m	The resulting handle. Only valid if this Builtin returns BI_SUCCESS.
<return>	int	O	m	One of the return codes specified in Table 367.

4.212 Builtin pop_abort_method**Purpose**

The Builtin pop_abort_method removes an abort method from the top of the abort method list, which is the list of methods to be executed if the current method is aborted.

Lexical structure

pop_abort_method(VOID)

The lexical element of the Builtin pop_abort_method is shown in Table 215.

Table 215 – Builtin pop_abort_method

Parameter name	Type	Direction	Usage	Description
<return>	int	O	m	One of the return codes specified in Table 367.

4.213 Builtin pow**Purpose**

The Builtin pow returns the power function of two floating-point values.

Lexical structure

pow(x, y)

The lexical elements of the Builtin pow are shown in Table 216.

Table 216 – Builtin pow

Parameter name	Type	Direction	Usage	Description
x	double	I	m	The base.
y	double	I	m	The exponent.
<return>	double	O	m	The value of x raised to the y power.

4.214 Builtin process_abort**Purpose**

The Builtin process_abort will abort the current method and execute any abort methods that 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 217.

Table 217 – Builtin process_abort

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.215 Builtin push_abort_method

Purpose

The Builtin push_abort_method adds an abort method to the top of the abort method list, which is the list of methods to be executed if the current method is aborted.

Lexical structure

push_abort_method(abort_method_id)

The lexical elements of the Builtin push_abort_method are shown in Table 218.

Table 218 – Builtin push_abort_method

Parameter name	Type	Direction	Usage	Description
abort_method_id	unsigned long	l	m	The abort method to be added.
<return>	int	o	m	One of the return codes specified in Table 367.

4.216 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 219.

Table 219 – Builtin put_date

Parameter name	Type	Direction	Usage	Description
data	char[]	l	m	The new value of the VARIABLE.
size	long	l	m	The length of the new value.
<return>	long	O	m	One of the return codes specified in Table 366.

4.217 Builtin put_date_value

Purpose

The Builtin put_date_value stores the provided value of the specified scaled variable into the cache 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 cache 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:

- `DATE_AND_TIME`
- `TIME`
- `DURATION`

The variable shall be valid in the cache 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 220.

Table 220 – Builtin `put_date_value`

Parameter name	Type	Direction	Usage	Description
<code>id</code>	unsigned long	I	o	The item ID of the VARIABLE to access.
<code>member_id</code>	unsigned long	I	o	The member ID of the VARIABLE access.
<code>data</code>	<code>char[]</code>	I	m	The new value of the VARIABLE.
<code>size</code>	long	I	m	The length of the new value.
<code><return></code>	long	O	m	One of the return codes specified in Table 366.

4.218 Builtin `put_date_value2`

Purpose

The Builtin `put_date_value2` stores the provided value of the specified scaled variable into the cache 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 cache 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 cache or device.

Lexical structure

`put_date_value2(blk_id, blk_num, id, member_id, data, size)`

The lexical elements of the Builtin put_date_value2 are shown in Table 221.

Table 221 – Builtin put_date_value2

Parameter name	Type	Direction	Usage	Description
blk_id	unsigned long	I	m	The item ID of the BLOCK_A instance to access.
blk_num	unsigned long	I	m	The occurrence number of the BLOCK_A instance to access.
id	unsigned long	I	o	The item ID of the VARIABLE to access.
member_id	unsigned long	I	o	The member ID of the VARIABLE access.
data	char[]	I	m	The new value of the VARIABLE.
size	long	I	m	The length of the new value.
<return>	long	O	m	One of the return codes specified in Table 366.

4.219 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 222.

Table 222 – Builtin put_double

Parameter name	Type	Direction	Usage	Description
value	double	I	m	The new value of the VARIABLE.
<return>	long	O	m	One of the return codes specified in Table 366.

4.220 Builtin put_double_value

Purpose

The Builtin put_double_value stores the provided value of the specified scaled variable into the cache or device.

The parameters id and member_id are specified using the Builtins ITEM_ID and MEMBER_ID and the name or reference of the variable in question.

The specified value of a double variable is put into the cache or device. The variable shall have a data type of double and be a valid variable in the cache or device.

Lexical structure

```
put_double_value(id, member_id, value)
```

The lexical elements of the Builtin `put_double_value` are shown in Table 223.

Table 223 – Builtin `put_double_value`

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	o	The item ID of the VARIABLE to access.
member_id	unsigned long	I	o	The member ID of the VARIABLE access.
value	double	I	m	The new value of the VARIABLE.
<return>	long	O	m	One of the return codes specified in Table 366.

4.221 Builtin `put_double_value2`

Purpose

The Builtin `put_double_value2` stores the provided value of the specified scaled variable into the cache 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 a double variable into the cache or device. The variable shall have a data type of double and be a valid variable in the cache or device.

Lexical structure

```
put_double_value2(blk_id, blk_num, id, member_id, value)
```

The lexical elements of the Builtin `put_double_value2` are shown in Table 224.

Table 224 – Builtin `put_double_value2`

Parameter name	Type	Direction	Usage	Description
blk_id	unsigned long	I	m	The item ID of the BLOCK_A instance to access.
blk_num	unsigned long	I	m	The occurrence number of the BLOCK_A instance to access.
id	unsigned long	I	o	The item ID of the VARIABLE to access.
member_id	unsigned long	I	o	The member ID of the VARIABLE access.
value	double	I	m	The new value of the VARIABLE.
<return>	long	O	m	One of the return codes specified in Table 366.

4.222 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 225.

Table 225 – Builtin `put_float`

Parameter name	Type	Direction	Usage	Description
value	double	I	m	The new value of the VARIABLE. The value is passed as a double to allow an expression to be passed. NOTE Many compilers perform a single to double conversion prior to evaluating expressions.
<return>	long	O	m	One of the return codes specified in Table 366.

4.223 Builtin `put_float_value`

Purpose

The Builtin `put_float_value` stores the provided value of the specified scaled variable in the cache 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 cache or device.

Lexical structure

`put_float_value(id, member_id, value)`

The lexical elements of the Builtin `put_float_value` are shown in Table 226.

Table 226 – Builtin `put_float_value`

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	o	The item ID of the VARIABLE to access.
member_id	unsigned long	I	o	The member ID of the VARIABLE access.
value	double	I	m	The new value of the VARIABLE.
<return>	long	O	m	One of the return codes specified in Table 366.

4.224 Builtin `put_float_value2`

Purpose

The Builtin `put_float_value2` stores the provided value of the specified scaled variable in the cache 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 cache or device.

Lexical structure

```
put_float_value2(blk_id, blk_num, id, member_id, value)
```

The lexical elements of the Builtin `put_float_value2` are shown in Table 227.

Table 227 – Builtin `put_float_value2`

Parameter name	Type	Direction	Usage	Description
<code>blk_id</code>	unsigned long	I	m	The item ID of the <code>BLOCK_A</code> instance to access.
<code>blk_num</code>	unsigned long	I	m	The occurrence number of the <code>BLOCK_A</code> instance to access.
<code>id</code>	unsigned long	I	o	The item ID of the <code>VARIABLE</code> to access.
<code>member_id</code>	unsigned long	I	o	The member ID of the <code>VARIABLE</code> access.
<code>value</code>	double	I	m	The new value of the <code>VARIABLE</code> .
<code><return></code>	long	O	m	One of the return codes specified in Table 366.

4.225 Builtin `PUT_MESSAGE`

Purpose

The Builtin `PUT_MESSAGE` will display the specified message on the screen. Any embedded local variable references or references to `VARIABLE`s will be expanded to the variable's value.

References to variables may be embedded on the message string allowing their value to be displayed using the following syntax:

```
%{variable-reference}
```

or

```
%[format]{variable-reference}
```

The optional format supports all `fprintf` format strings in the programming language-C (see ISO/IEC 9899) plus the additions shown in Table 228.

Table 228 – Format options

Format	Description
D	The referenced item is of CLASS DYNAMIC and the display shall be updated continuously while the method execution is blocked by the Builtin function.
L	The LABEL of the referenced item shall be displayed.
U	The unit of the referenced item shall be displayed.

If more than one format is given, the format specifiers shall be separated by a comma.

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 229.

Table 229 – Builtin PUT_MESSAGE

Parameter name	Type	Direction	Usage	Description
message	char[]	I	m	A message to be displayed.
<return>	VOID	—	—	—

4.226 Builtin put_message

Purpose

The Builtin put_message will display the specified message on the screen. Any embedded variable references will be expanded to the current value of the variable. ENUMERATED variables are expanded to their textual representation rather than their numerical representation.

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.

The following syntax is used to reference an VARIABLE from the list of VARIABLES given in the parameter global_var_ids:

%{global-var-ids-array-index}

or

%[format]{global-var-ids-array-index}

The optional format supports all fprintf format strings the programming language-C (see ISO/IEC 9899) plus the additions shown in Table 228.

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 230.

Table 230 – Builtin put_message

Parameter name	Type	Direction	Usage	Description
message	char[]	I	m	A message to be displayed.
global_var_ids	int[]	I	m	An array which contains unique identifiers of VARIABLE instance.
<return>	VOID	—	—	—

4.227 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 231.

Table 231 – Builtin put_signed

Parameter name	Type	Direction	Usage	Description
value	long	I	m	The new value of the VARIABLE.
<return>	long	O	m	One of the return codes specified in Table 366.

4.228 Builtin put_signed_value

Purpose

The Builtin put_signed_value stores the new value of the specified scaled variable into the cache 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 a signed integer variable into the cache or device.

The variable to be written shall have a long data type and be a valid variable in the cache or device.

Lexical structure

put_signed_value(id, member_id, value)

The lexical elements of the Builtin put_signed_value are shown in Table 232.

Table 232 – Builtin put_signed_value

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	o	The item ID of the VARIABLE to access.
member_id	unsigned long	I	o	The member ID of the VARIABLE access.
value	long	I	m	The new value of the VARIABLE.
<return>	long	O	m	One of the return codes specified in Table 366.

4.229 Builtin put_signed_value2

Purpose

The Builtin put_signed_value2 stores the new value of the specified scaled variable into the cache 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 a signed integer variable into the cache or device.

The variable to be written shall have a long data type and be a valid variable in the cache or device.

Lexical structure

put_signed_value2(blk_id, blk_num, id, member_id, value)

The lexical elements of the Builtin put_signed_value2 are shown in Table 233.

Table 233 – Builtin put_signed_value2

Parameter name	Type	Direction	Usage	Description
blk_id	unsigned long	I	m	The item ID of the BLOCK_A instance to access.
blk_num	unsigned long	I	m	The occurrence number of the BLOCK_A instance to access.
id	unsigned long	I	o	The item ID of the VARIABLE to access.
member_id	unsigned long	I	o	The member ID of the VARIABLE access.
value	long	I	m	The new value of the VARIABLE.
<return>	long	O	m	One of the return codes specified in Table 366.

4.230 Builtin put_string

Purpose

The Builtin put_string stores the new string value of the variable being operated on.

The variable shall be valid in the cache 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 unicode)
- BITSTRING

Device variables with data types of enumerated, bit_enumerated, and index shall be handled by the method because they are not supported by the programming language-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 234.

Table 234 – Builtin `put_string`

Parameter name	Type	Direction	Usage	Description
string	char[]	I	m	The new value of the VARIABLE.
len	long	I	m	The length of the new value.
<return>	long	O	m	One of the return codes specified in Table 366.

4.231 Builtin `put_string_value`

Purpose

The Builtin `put_string_value` stores the provided value of the specified scaled variable into the cache 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 cache 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 unicode)
- BITSTRING

Device variables with data types of enumerated, bit_enumerated, and index shall be handled by the method because they are not supported by the programming language-C.

Lexical structure

```
put_string_value(id, member_id, string, len)
```

The lexical elements of the Builtin `put_string_value` are shown in Table 235.

Table 235 – Builtin `put_string_value`

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	o	The item ID of the VARIABLE to access.
member_id	unsigned long	I	o	The member ID of the VARIABLE access.
string	char[]	I	m	The new value of the VARIABLE.
len	long	I	m	The length of the new value.
<return>	long	O	m	One of the return codes specified in Table 366.

4.232 Builtin `put_string_value2`

Purpose

The Builtin `put_string_value2` stores the provided value of the specified scaled variable into the cache 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 cache 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 unicode)
- BITSTRING

Device variables with data types of `enumerated`, `bit_enumerated`, and `index` shall be handled by the method because they are not supported by programming language-C.

Lexical structure

```
put_string_value2(blk_id, blk_num, id, member_id, string, len)
```

The lexical elements of the Builtin `put_string_value2` are shown in Table 236.

Table 236 – Builtin `put_string_value2`

Parameter name	Type	Direction	Usage	Description
<code>blk_id</code>	unsigned long	I	m	The item ID of the <code>BLOCK_A</code> instance to access.
<code>blk_num</code>	unsigned long	I	m	The occurrence number of the <code>BLOCK_A</code> instance to access.
<code>id</code>	unsigned long	I	o	The item ID of the <code>VARIABLE</code> to access.
<code>member_id</code>	unsigned long	I	o	The member ID of the <code>VARIABLE</code> access.
<code>string</code>	<code>char[]</code>	I	m	The new value of the <code>VARIABLE</code> .
<code>len</code>	long	I	m	The length of the new value.
<code><return></code>	long	O	m	One of the return codes specified in Table 366.

4.233 Builtin `put_unsigned`

Purpose

The Builtin `put_unsigned` stores the new unscaled unsigned value of the variable being operated on. The variable shall have a data type of `unsigned` and be a valid variable in the cache or device. An unsigned value may contain device variables of the following data types:

- UNSIGNED
- ENUMERATED
- BIT_ENUMERATED
- INDEX

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 237.

Table 237 – Builtin put_unsigned

Parameter name	Type	Direction	Usage	Description
value	unsigned long	I	o	The new value of the VARIABLE.
<return>	long	O	m	One of the return codes specified in Table 366.

4.234 Builtin put_unsigned_value**Purpose**

The Builtin put_unsigned_value stores the provided value of the specified scaled variable into the cache 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 cache 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 the programming language-C. The variable shall be valid in the cache or device.

Lexical structure

```
put_unsigned_value(id, member_id, value)
```

The lexical elements of the Builtin put_unsigned_value are shown in Table 238.

Table 238 – Builtin put_unsigned_value

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	o	The item ID of the VARIABLE to access.
member_id	unsigned long	I	o	The member ID of the VARIABLE access.
value	unsigned long	I	o	The new value of the VARIABLE.
<return>	long	O	m	One of the return codes specified in Table 366.

4.235 Builtin put_unsigned_value2**Purpose**

The Builtin put_unsigned_value2 stores the provided value of the specified scaled variable into the cache 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 cache 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 the programming language-C. The variable shall be valid in the cache or device.

Lexical structure

`put_unsigned_value2(blk_id, blk_num, id, member_id, value)`

The lexical elements of the Builtin `put_unsigned_value2` are shown in Table 239.

Table 239 – Builtin `put_unsigned_value2`

Parameter name	Type	Direction	Usage	Description
<code>blk_id</code>	unsigned long	I	m	The item ID of the BLOCK_A instance to access.
<code>blk_num</code>	unsigned long	I	m	The occurrence number of the BLOCK_A instance to access.
<code>id</code>	unsigned long	I	o	The item ID of the VARIABLE to access.
<code>member_id</code>	unsigned long	I	o	The member ID of the VARIABLE access.
<code>value</code>	unsigned long	I	o	The new value of the VARIABLE.
<return>	long	O	m	One of the return codes specified in Table 366.

4.236 Builtin `re_read_file`

Purpose

The Builtin `re_read_file` causes the FILE to be reread.

Lexical structure

`re_read_file(file_item)`

The lexical elements of the Builtin `re_read_file` are shown in Table 240.

Table 240 – Builtin `re_read_file`

Parameter name	Type	Direction	Usage	Description
<code>file_item</code>	DD_ITEM &	I	m	The FILE to read.
<return>	int	O	m	One of the return codes specified in Table 367.

4.237 Builtin `re_write_file`

Purpose

The Builtin `re_write_file` causes the FILE to be rewritten.

Lexical structure

```
re_write_file(file_item)
```

The lexical elements of the Builtin `re_write_file` are shown in Table 241.

Table 241 – Builtin `re_write_file`

Parameter name	Type	Direction	Usage	Description
file_item	DD_ITEM &	I	m	The FILE to write.
<return>	int	O	m	One of the return codes specified in Table 367.

4.238 Builtin `read_value`

Purpose

The Builtin `read_value` causes the value of the parameter to be read from the device.

The Builtin `read_value` does nothing in a non-caching EDD application because a non-caching EDD application does not have a cache to update.

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 242.

Table 242 – Builtin `read_value`

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	o	The item ID of the VARIABLE to access.
member_id	unsigned long	I	o	The member ID of the VARIABLE access.
<return>	long	O	m	One of the return codes specified in Table 366.

4.239 Builtin `read_value2`

Purpose

The Builtin `read_value2` causes the parameter's value to be read from the device.

The Builtin `read_value` does nothing in a non-caching EDD application because a non-caching EDD application does not have a cache to update.

The parameters `id` and `member_id` are specified using the Builtins `ITEM_ID` and `MEMBER_ID` and the name or reference of the variable in question.

Lexical structure

```
read_value2(blk_id, blk_num, id, member_id)
```

The lexical elements of the Builtin `read_value2` are shown in Table 243.

Table 243 – Builtin read_value2

Parameter name	Type	Direction	Usage	Description
blk_id	unsigned long	I	m	The item ID of the BLOCK_A instance to access.
blk_num	unsigned long	I	m	The occurrence number of the BLOCK_A instance to access.
id	unsigned long	I	o	The item ID of the VARIABLE to access.
member_id	unsigned long	I	o	The member ID of the VARIABLE access.
<return>	long	O	m	One of the return codes specified in Table 366.

4.240 Builtin ReadCommand**Purpose**

The Builtin ReadCommand allows reading variables with a given command.

Lexical structure

`ReadCommand(command_identifier)`

The lexical elements of the Builtin ReadCommand are shown in Table 244.

Table 244 – Builtin ReadCommand

Parameter name	Type	Direction	Usage	Description
command_identifier	reference	I	m	The ID of the command to execute.
<return>	long	O	m	A protocol specific error value, the encoding of which is defined by the specific protocol or device.

4.241 Builtin readItemFromDevice**Purpose**

This Builtin reads an item from an open block transfer indicated by the handle.

The EDD_ITEM may be a compound item but shall contain only data.

Lexical structure

`readItemFromDevice(handle, dd_item)`

The lexical elements of the Builtin readItemFromDevice are shown in Table 245.

Table 245 – Builtin readItemFromDevice

Parameter name	Type	Direction	Usage	Description
handle	unsigned int	I	m	The handle of the block transfer.
dd_item	DD_ITEM &	O	m	The DD item to read.
<return>	int	O	m	One of the return codes specified in Table 367.

4.242 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 shall 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 246.

Table 246 – Builtin remove_abort_method

Parameter name	Type	Direction	Usage	Description
abort_method_name	reference	I	m	The ID of the method.
<return>	int	O	m	One of the return codes specified in Table 367.

4.243 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 247.

Table 247 – Builtin remove_abort_method

Parameter name	Type	Direction	Usage	Description
method_id	unsigned long	I	m	The ID of the method.
<return>	long	O	m	One of the return codes specified in Table 366.

4.244 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 248.

Table 248 – Builtin `remove_all_abort_methods`

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.245 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 249.

Table 249 – Builtin `resolve_array_ref`

Parameter name	Type	Direction	Usage	Description
<code>id</code>	unsigned long	I	m	The item ID of the <code>REFERENCE_ARRAY</code> .
<code>member_id</code>	unsigned long	I	m	The index of the requested <code>REFERENCE_ARRAY</code> element.
<return>	unsigned long	O	m	The item ID of the resolved reference, or zero for an error. In the case of an error, Builtin <code>get_resolve_status</code> may be used to access the actual error code.

4.246 Builtin `resolve_array_ref2`

Purpose

The Builtin `resolve_array_ref2` is used by method developers when parsing things such as `item_array_name`. The Builtin `resolve_array_ref2` takes the item ID of the `REFERENCE_ARRAY` and an index and resolves them into the item ID of the `VALUE_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 VALUE_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_ref2(block_id, block_instance, id, member_id)
```

The lexical elements of the Builtin resolve_array_ref2 are shown in Table 250.

Table 250 – Builtin resolve_array_ref2

Parameter name	Type	Direction	Usage	Description
block_id	unsigned long	I	m	The item ID of the BLOCK_A being resolved.
block_instance	unsigned long	I	m	The occurrence number of the BLOCK_A being resolved.
id	unsigned long	I	m	The item ID of the VALUE_ARRAY.
member_id	unsigned long	I	m	The index of the requested VALUE_ARRAY element.
<return>	unsigned long	O	m	The item ID of the resolved reference, or zero for an error. In the case of an error, Builtin get_resolve_status may be used to access the actual error code.

4.247 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 251.

Table 251 – Builtin resolve_block_ref

Parameter name	Type	Direction	Usage	Description
member_id	unsigned long	I	m	The member ID of BLOCK_A.
<return>	unsigned long	O	m	The item ID of the resolved reference, or zero for an error. In the case of an error, Builtin get_resolve_status may be used to access the actual error code.

4.248 Builtin resolve_block_ref2

Purpose

The Builtin resolve_block_ref2 is used by method developers when parsing things such as BLOCK_A[block_instance].name. The Builtin resolve_block_ref2 takes the block_instance index and 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_ref2(block_id, block_instance, member_id)

The lexical elements of the Builtin resolve_block_ref2 are shown in Table 252.

Table 252 – Builtin resolve_block_ref2

Parameter name	Type	Direction	Usage	Description
block_id	unsigned long	I	m	The item ID of the BLOCK_A being resolved.
block_instance	unsigned long	I	m	The occurrence number of the BLOCK_A being resolved.
member_id	unsigned long	I	m	The member ID of BLOCK_A.
<return>	unsigned long	O	m	The item ID of the resolved reference, or zero for an error. In the case of an error, Builtin get_resolve_status may be used to access the actual error code.

4.249 Builtin resolve_list_ref

Purpose

The Builtin resolve_list_ref returns the type of a LIST.

Lexical structure

resolve_list_ref(list_id)

The lexical elements of the Builtin resolve_list_ref are shown in Table 253.

Table 253 – Builtin resolve_list_ref

Parameter name	Type	Direction	Usage	Description
list_id	unsigned long	I	m	The member ID of the LIST.
<return>	unsigned long	O	m	The item ID of the resolved reference, or zero for an error. In the case of an error, Builtin get_resolve_status may be used to access the actual error code.

4.250 Builtin resolve_local_ref

Purpose

The Builtin `resolve_local_ref` is used by method developers when parsing things such as `LOCAL_PARAM.name`. It takes the member ID of a `LOCAL_PARAMETERS` element and returns the item ID of the completely resolved element.

The parameter `member_id` is specified using the `MEMBER_ID` Builtin and the name or reference of the variable in question. It resolves the `member_id` reference generated for a `LOCAL_PARAMETERS` element into an item ID, which is then used in Builtin calls.

Lexical structure

`resolve_local_ref(member_id)`

The lexical elements of the Builtin `resolve_local_ref` are shown in Table 254.

Table 254 – Builtin resolve_local_ref

Parameter name	Type	Direction	Usage	Description
<code>member_id</code>	unsigned long	I	m	The member ID of the local parameter.
<code><return></code>	unsigned long	O	m	The item ID of the resolved reference, or zero for an error. In the case of an error, Builtin <code>get_resolve_status</code> may be used to access the actual error code.

4.251 Builtin resolve_local_ref2

Purpose

The Builtin `resolve_local_ref2` is used by method developers when parsing things such as `block_name[n].LOCAL_PARAM.name`. It takes the member ID of a `BLOCK_A`, the occurrence number of a `BLOCK_A`, the member ID of a `LOCAL_PARAMETERS` element and returns the item ID of the completely resolved element.

The parameters `block_id` and `member_id` are specified using the `MEMBER_ID` Builtin. It resolves the `member_id` reference generated for a `LOCAL_PARAMETERS` element into an item ID, which is then used in Builtin calls.

Lexical structure

`resolve_local_ref2(block_id, block_instance, member_id)`

The lexical elements of the Builtin `resolve_local_ref2` are shown in Table 255.

Table 255 – Builtin resolve_local_ref2

Parameter name	Type	Direction	Usage	Description
<code>block_id</code>	unsigned long	I	m	The item ID of the <code>BLOCK_A</code> being resolved.
<code>block_instance</code>	unsigned long	I	m	The occurrence number of the <code>BLOCK_A</code> being resolved.
<code>member_id</code>	unsigned long	I	m	The member ID of the local parameter.
<code><return></code>	unsigned long	O	m	The item ID of the resolved reference, or zero for an error. In the case of an error, Builtin <code>get_resolve_status</code> may be used to access the actual error code.

4.252 Builtin resolve_param_list_ref

Purpose

The Builtin resolve_param_list_ref is used by method developers when parsing things such as a 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 256.

Table 256 – Builtin resolve_param_list_ref

Parameter name	Type	Direction	Usage	Description
member_id	unsigned long	I	m	The member ID of the parameter list.
<return>	unsigned long	O	m	The item ID of the resolved reference, or zero for an error. In the case of an error, Builtin get_resolve_status may be used to access the actual error code.

4.253 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 257.

Table 257 – Builtin resolve_param_ref

Parameter name	Type	Direction	Usage	Description
member_id	unsigned long	I	m	The member ID of the parameter.
<return>	unsigned long	O	m	The item ID of the resolved reference, or zero for an error. In the case of an error, Builtin get_resolve_status may be used to access the actual error code.

4.254 Builtin resolve_param_ref2

Purpose

The Builtin resolve_param_ref2 is used by method developers when parsing things such as block_name[n].PARAM.name. It takes the member ID of a BLOCK_A, an occurrence number, and a PARAMETER element and returns the item ID of the completely resolved element.

The block name and parameter member_id are specified using the MEMBER_ID Builtin and the name or reference of the variable in question. It resolves the member_id reference generated for a PARAMETER element into an item ID, which is then used in Builtin calls.

Lexical structure

resolve_param_ref2(block_id, block_instance, member_id)

The lexical elements of the Builtin resolve_param_ref2 are shown in Table 258.

Table 258 – Builtin resolve_param_ref2

Parameter name	Type	Direction	Usage	Description
block_id	unsigned long	I	m	The item ID of the BLOCK_A being resolved.
block_instance	unsigned long	I	m	The occurrence number of the BLOCK_A being resolved.
member_id	unsigned long	I	m	The member ID of the parameter.
<return>	unsigned long	O	m	The item ID of the resolved reference, or zero for an error. In the case of an error, Builtin get_resolve_status may be used to access the actual error code.

4.255 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 259.

Table 259 – Builtin resolve_record_ref

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	m	The item ID of the RECORD being resolved.
member_id	unsigned long	I	m	The member ID of the record member.
<return>	unsigned long	O	m	The item ID of the resolved reference, or zero for an error. In the case of an error, Builtin get_resolve_status may be used to access the actual error code.

4.256 Builtin resolve_record_ref2**Purpose**

The Builtin resolve_record_ref2 is used by method developers when parsing things such as record_name.member_name of a specific block. It takes the item ID of a block, the occurrence number of a block, the item ID of a record and the member ID of a record element or member and returns the item ID of the completely resolved element.

If a member of a record has a conditional dependency (that is, the definition of an element is dependent on another value), the dependencies are taken into account during the resolution.

The parameters block_id, id and member_id are specified using the Bultins ITEM_ID and MEMBER_ID and the name or reference of the variable in question.

Resolves the member_id reference generated for a RECORD element into an item ID, which is then used in Builtin calls.

Lexical structure

```
resolve_record_ref2(block_id, block_instance, id, member_id)
```

The lexical elements of the Builtin resolve_record_ref2 are shown in Table 260.

Table 260 – Builtin resolve_record_ref2

Parameter name	Type	Direction	Usage	Description
block_id	unsigned long	I	m	The item ID of the BLOCK_A being resolved.
block_instance	unsigned long	I	m	The occurrence number of the BLOCK_A being resolved.
id	unsigned long	I	m	The item ID of the RECORD being resolved.
member_id	unsigned long	I	m	The member ID of the record member.
<return>	unsigned long	O	m	The item ID of the resolved reference, or zero for an error. In the case of an error, Builtin get_resolve_status may be used to access the actual error code.

4.257 Builtin ret_double_value**Purpose**

The Builtin ret_double_value calls the Builtin get_double_value and returns the value.

Lexical structure

```
ret_double_value(item_id, member_id)
```

The lexical elements of the Builtin `ret_double_value` are shown in Table 261.

Table 261 – Builtin `ret_double_value`

Parameter name	Type	Direction	Usage	Description
item_id	unsigned long	I	m	The item ID of the VARIABLE.
member_id	unsigned long	I	m	The member ID of the VARIABLE.
<return>	double	O	m	The specified value.

4.258 Builtin `ret_double_value2`**Purpose**

The Builtin `ret_double_value2` calls the Builtin `get_double_value2` and returns the value.

Lexical structure

```
ret_double_value2(block_id, block_instance, item_id, member_id)
```

The lexical elements of the Builtin `ret_double_value2` are shown in Table 262.

Table 262 – Builtin `ret_double_value2`

Parameter name	Type	Direction	Usage	Description
block_id	unsigned long	I	m	The item ID of the block to be accessed.
block_instance	unsigned long	I	m	The occurrence number of the block to be accessed.
item_id	unsigned long	I	m	The item ID of the VARIABLE.
member_id	unsigned long	I	m	The member ID of the VARIABLE.
<return>	double	O	m	The specified value.

4.259 Builtin `ret_float_value`**Purpose**

The Builtin `ret_float_value` calls the Builtin `get_float_value` and returns the value.

Lexical structure

```
ret_float_value(item_id, member_id)
```

The lexical elements of the Builtin `ret_float_value` are shown in Table 263.

Table 263 – Builtin ret_float_value

Parameter name	Type	Direction	Usage	Description
item_id	unsigned long	I	m	The item ID of the VARIABLE.
member_id	unsigned long	I	m	The member ID of the VARIABLE.
<return>	float	O	m	The specified value.

4.260 Builtin ret_float_value2**Purpose**

The Builtin ret_float_value2 calls the Builtin get_float_value2 and returns the value.

Lexical structure

```
ret_float_value2(block_id, block_instance, item_id, member_id)
```

The lexical elements of the Builtin ret_float_value2 are shown in Table 264.

Table 264 – Builtin ret_float_value2

Parameter name	Type	Direction	Usage	Description
block_id	unsigned long	I	m	The item ID of the block to be accessed.
block_instance	unsigned long	I	m	The occurrence number of the block to be accessed.
item_id	unsigned long	I	m	The item ID of the VARIABLE.
member_id	unsigned long	I	m	The member ID of the VARIABLE.
<return>	float	O	m	The specified value.

4.261 Builtin ret_signed_value**Purpose**

The Builtin ret_signed_value calls the Builtin get_signed_value and returns the value.

Lexical structure

```
ret_signed_value(item_id, member_id)
```

The lexical elements of the Builtin ret_signed_value are shown in Table 265.

Table 265 – Builtin ret_signed_value

Parameter name	Type	Direction	Usage	Description
item_id	unsigned long	I	m	The item ID of the VARIABLE.
member_id	unsigned long	I	m	The member ID of the VARIABLE.
<return>	long	O	m	The specified value.

4.262 Builtin ret_signed_value2

Purpose

The Builtin ret_signed_value2 calls the Builtin get_signed_value2 and returns the value.

Lexical structure

ret_signed_value2(block_id, block_instance, item_id, member_id)

The lexical elements of the Builtin ret_signed_value2 are shown in Table 266.

Table 266 – Builtin ret_signed_value2

Parameter name	Type	Direction	Usage	Description
block_id	unsigned long	I	m	The item ID of the block to be accessed.
block_instance	unsigned long	I	m	The occurrence number of the block to be accessed.
item_id	unsigned long	I	m	The item ID of the VARIABLE.
member_id	unsigned long	I	m	The member ID of the VARIABLE.
<return>	long	O	m	The specified value.

4.263 Builtin ret_unsigned_value

Purpose

The Builtin ret_unsigned_value calls the Builtin get_unsigned_value and returns the value.

Lexical structure

ret_unsigned_value(item_id, member_id)

The lexical elements of the Builtin ret_unsigned_value are shown in Table 267.

Table 267 – Builtin ret_unsigned_value

Parameter name	Type	Direction	Usage	Description
item_id	unsigned long	I	m	The item ID of the VARIABLE.
member_id	unsigned long	I	m	The member ID of the VARIABLE.
<return>	unsigned long	O	m	The specified value.

4.264 Builtin ret_unsigned_value2

Purpose

The Builtin ret_unsigned_value2 calls the Builtin get_unsigned_value2 and returns the value.

Lexical structure

ret_unsigned_value2(block_id, block_instance, item_id, member_id)

The lexical elements of the Builtin ret_unsigned_value2 are shown in Table 268.

Table 268 – Builtin ret_unsigned_value2

Parameter name	Type	Direction	Usage	Description
block_id	unsigned long	I	m	The item ID of the block to be accessed.
block_instance	unsigned long	I	m	The occurrence number of the block to be accessed.
item_id	unsigned long	I	m	The item ID of the VARIABLE.
member_id	unsigned long	I	m	The member ID of the VARIABLE.
<return>	unsigned long	O	m	The specified value.

4.265 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 profiles 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 269.

Table 269 – Builtin retry_on_all_comm_errors

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.266 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 270.

Table 270 – Builtin RETRY_ON_ALL_COMM_STATUS

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.267 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 271.

Table 271 – Builtin RETRY_ON_ALL_DEVICE_STATUS

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.268 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 272.

Table 272 – Builtin RETRY_ON_ALL_RESPONSE_CODES

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.269 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 273.

Table 273 – Builtin retry_on_all_response_codes

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.270 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 274.

Table 274 – Builtin RETRY_ON_COMM_ERROR

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.271 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 275.

Table 275 – Builtin retry_on_comm_error

Parameter name	Type	Direction	Usage	Description
error	unsigned long	I	m	The communication errors.
<return>	long	O	m	One of the return codes specified in Table 366.

4.272 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 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_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 276.

Table 276 – Builtin RETRY_ON_COMM_STATUS

Parameter name	Type	Direction	Usage	Description
comm_status	int	l	m	The new communication status retry mask.
<return>	VOID	—	—	—

4.273 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 277.

Table 277 – Builtin RETRY_ON_DEVICE_STATUS

Parameter name	Type	Direction	Usage	Description
device_status	int	l	m	The new communication status retry mask.
<return>	VOID	—	—	—

4.274 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 278.

Table 278 – Builtin RETRY_ON_NO_DEVICE

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.275 Builtin RETRY_ON_RESPONSE_CODE

Purpose

The Builtin RETRY_ON_RESPONSE_CODE will set the correct bit(s) in the response code retry mask such that the specified response code value will cause the current command to be retried. The response code is defined to be the first data octet returned in a transaction, when bit 7 of this octet is 0.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of available masks and their default values.

The mask affected by this function is used by the Builtins send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status and display.

Lexical structure

RETRY_ON_RESPONSE_CODE(response_code)

The lexical elements of the Builtin RETRY_ON_RESPONSE_CODE are shown in Table 279.

Table 279 – Builtin RETRY_ON_RESPONSE_CODE

Parameter name	Type	Direction	Usage	Description
response_code	int	—	m	The new communication status retry mask.
<return>	VOID	—	—	—

4.276 Builtin retry_on_response_code

Purpose

The Builtin retry_on_response_code causes the last request made by a method to be retried three times after a specific response code is received.

This Builtin frees a method from having to explicitly retry a request after a given response code is received.

A response code is a value representing an application-specific error condition.

NOTE A response code value of zero is not considered an error response code. Each response code and response code type is defined in the EDD.

Retries the method's last request upon receiving the specified response code.

Lexical structure

retry_on_response_code(code)

The lexical elements of the Builtin retry_on_response_code are shown in Table 280.

Table 280 – Builtin retry_on_response_code

Parameter name	Type	Direction	Usage	Description
code	unsigned long	I	m	The response code being resolved.
<return>	long	O	m	One of the return codes specified in Table 366.

4.277 Builtin round**Purpose**

The Builtin round rounds a floating-point value to the nearest integer.

Lexical structure

`round(x)`

The lexical elements of the Builtin round are shown in Table 281.

Table 281 – Builtin round

Parameter name	Type	Direction	Usage	Description
x	double	I	m	A floating-point value.
<return>	double	O	m	The rounded value.

4.278 Builtin save_on_exit**Purpose**

The Builtin save_on_exit sets the termination action to save any changes made by the method to variables in the cache not already sent to the device. The values in the device are not affected. As long as the application has not changed the values, the values are saved as if written by an application.

When a method modifies a variable's value, it is really just changing a resident copy of that value in the EDD application. This change is effective only for the life of the method. To make the values permanent in the cache, the method shall save the values before exiting.

All values that have been changed, but not sent to the device, are handled at the termination of the method by either sending them to the device, discarding them, or saving the value changes in a way that survives the end of the method (but does not affect the device). Saved values can be used by other methods.

If a method does not call Builtin discard_on_exit, save_on_exit, or send_on_exit prior to exiting, any values in the Builtin are discarded as if discard_on_exit was called.

If the method aborts, then the Builtin discard_on_exit option is enforced even if Builtin save_on_exit was called before the method aborted.

Lexical structure

`save_on_exit(VOID)`

The lexical element of the Builtin save_on_exit is shown in Table 282.

Table 282 – Builtin save_on_exit

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.279 Builtin save_values**Purpose**

The Builtin save_values will cause the values of all device variables modified during a method session to remain permanent after the method has been exited.

Device variables that are modified, but are not sent to the connected device, will be restored to their previous value if this function is not called.

Lexical structure

save_values(VOID)

The lexical element of the Builtin save_values is shown in Table 283.

Table 283 – Builtin save_values

Parameter name	Type	Direction	Usage	Description
<return>	int	O	m	One of the return codes specified in Table 367.

4.280 Builtin seconds_to_TIME_VALUE**Purpose**

The Builtin seconds_to_TIME_VALUE converts a seconds value to a TIME_VALUE.

Lexical structure

seconds_to_TIME_VALUE(seconds)

The lexical element of the Builtin seconds_to_TIME_VALUE is shown in Table 284.

Table 284 – Builtin seconds_to_TIME_VALUE

Parameter name	Type	Direction	Usage	Description
seconds	double	I	m	The number of seconds.
<return>	unsigned long	O	m	The TIME_VALUE(4) value.

4.281 Builtin seconds_to_TIME_VALUE8**Purpose**

The Builtin seconds_to_TIME_VALUE8 converts a seconds value to a TIME_VALUE(8).

Lexical structure

seconds_to_TIME_VALUE8(seconds)

The lexical element of the Builtin seconds_to_TIME_VALUE is shown in Table 285.

Table 285 – Builtin seconds_to_TIME_VALUE

Parameter name	Type	Direction	Usage	Description
seconds	double	I	m	The number of seconds.
<return>	unsigned long long	O	m	The TIME_VALUE(8) value.

4.282 Builtin SELECT_FROM_LIST**Purpose**

The Builtin SELECT_FROM_LIST has the same functionality as the Builtin SELECT_FROM_LIST, except that device variables are not allowed in the prompt string. There shall be at least two options in the select list.

NOTE 1 Previously displayed messages are not guaranteed to remain on the screen.

NOTE 2 Some EDD applications will clear the display upon each call to this function.

Lexical structure

SELECT_FROM_LIST(prompt, option_list)

The lexical elements of the Builtin SELECT_FROM_LIST are shown in Table 286.

Table 286 – Builtin SELECT_FROM_LIST

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	A message to be displayed.
option_list	char[]	I	m	The option string, consisting of separated text, that defines the menu items.
<return>	int	O	m	The result of the selection as position (zero based).

4.283 Builtin select_from_list**Purpose**

The Builtin select_from_list will display a prompt and a supplied list of options for the user to select from. Once a selection is made, the position of that option in the option list is returned. The position in the list is zero-based. For example, if the list provided contains two items and if the first item is selected, this function will return a 0. If the second item is selected, a 1 is returned.

The prompt and/or list may also contain embedded local and/or device variable values (see put_message for the lexical structure).

The options are passed to the function as a single string with semicolon separators between the options. There shall be at least two options in the select list.

NOTE 1 Previously displayed messages are not guaranteed to remain on the screen.

NOTE 2 Some EDD applications will clear the display upon each call to this function.

Lexical structure

select_from_list(prompt, global_var_ids, option_list)

The lexical elements of the Builtin select_from_list are shown in Table 287.

Table 287 – Builtin select_from_list

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	A message to be displayed.
global_var_ids	array of int	I	m	An array which contains unique identifiers of VARIABLE instances.
option_list	char[]	I	m	The option string, consisting of semicolon separated text, that defines the menu items.
<return>	int	O	m	The result of the selection as position (zero based).

4.284 Builtin select_from_menu**Purpose**

The Builtin select_from_menu displays a list of options or menu items for the user to select from and retrieves the selection number. When the user has made a choice it is returned to the calling method. Selection numbers start at 1 and range up to the number of options specified by the parameter options. Variable values may be embedded in the prompt message string with formatting information. The option string defines the menu from which the user selects and is a string consisting of the selection options; each option is separated from the other options with a semi-colon. Each option may have language codes embedded in it.

The application displays the strings in accordance with the following guidelines:

- the menu items appear on the screen in the same order as listed in the string;
- the prompt string appears at the top of the menu display on the screen;
- the prompt string is not selectable;
- the menu is presented vertically.

Lexical structure

```
select_from_menu(prompt, ids, indices, id_count, options, selection)
```

The lexical elements of the Builtin select_from_menu are shown in Table 288.

Table 288 – Builtin select_from_menu

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	A message to be displayed.
ids	array of unsigned long	I	m	The item IDs of the VARIABLEs used in the prompt message.
indices	array of unsigned long	I	m	The member IDs or indices of the VARIABLEs used in the prompt message.
id_count	long	I	m	The number of REFERENCE_ARRAY elements in the REFERENCE_ARRAY for IDs and indices. Set to zero if the REFERENCE_ARRAY for IDs and indices are zero.
options	char[]	I	m	The option string, consisting of separated text, that defines the menu items.
selection	long	O	m	The result of the selection as position (zero based).
<return>	long	O	m	One of the return codes specified in Table 366.

4.285 Builtin select_from_menu2

Purpose

The Builtin select_from_menu2 displays a list of options or menu items for the user to select from and retrieves the selection number. When the user has made a choice it is returned to the calling method. Selection numbers start at 1 and range up to the number of options specified by the parameter options. Variable values may be embedded in the prompt message string with formatting information. The option string defines the menu from which the user selects and is a string consisting of the selection options; each option is separated from the other options with a semi-colon. Each option may have language codes embedded in it.

The application displays the strings in accordance with the following guidelines:

- the menu items appear on the screen in the same order as listed in the string;
- the prompt string appears at the top of the menu display on the screen;
- the prompt string is not selectable;
- the menu is presented vertically.

Lexical structure

```
select_from_menu2(prompt, blk_ids, blk_nums, ids, indices, id_count,
options, selection)
```

The lexical elements of the Builtin select_from_menu2 are shown in Table 289.

Table 289 – Builtin select_from_menu2

Parameter name	Type	Direction	Usage	Description
prompt	char[]	I	m	A message to be displayed.
blk_ids	array of unsigned long	I	m	The item IDs of the BLOCK_A instances used in the prompt message.
blk_nums	array of unsigned long	I	m	The occurrence numbers of the BLOCK_A instances used in the prompt message.
ids	array of unsigned long	I	m	The item IDs of the VARIABLES used in the prompt message.
indices	array of unsigned long	I	m	The member IDs or indices of the VARIABLES used in the prompt message.
id_count	long	I	m	The number of REFERENCE_ARRAY elements in the REFERENCE_ARRAY for IDs and indices. Set to zero if the REFERENCE_ARRAY for IDs and indices are zero.
options	char[]	I	m	The option string, consisting of separated text, that defines the menu items.
selection	long	O	m	The result of the selection as position (zero based).
<return>	long	O	m	One of the return codes specified in Table 366.

4.286 Builtin send

Purpose

The Builtin send will send the specified command and return the response code, communications status, and command status respectively in the cmd_status array.

The octets in the cmd_status array are logically AND with the appropriate abort masks to determine if the method should be aborted. If there is a match, the method is aborted and the abort methods (if any were designated) are run. If there is no match for the abort masks, the same data is AND with the appropriate retry masks. If there is a match, the command will be retried. This continues until either the command succeeds, an abort occurs, or the maximum number of retries is reached.

The status octets for the command are returned in the cmd_status parameter, a pointer to a 3-octet array, which is allocated in the method making the call.

Lexical structure

send(cmd_number, cmd_status)

The lexical elements of the Builtin send are shown in Table 290.

Table 290 – Builtin send

Parameter name	Type	Direction	Usage	Description
cmd_number	int	I	m	The number of command to send.
cmd_status	char*	O	m	The status octets of the send command.
<return>	int	O	m	One of the return codes specified in Table 367.

4.287 Builtin send_all_values

Purpose

The Builtin send_all_values sends all of the values in the cache that the method modified but has not previously sent to the device. This action changes the device values to match the values set by the method.

This Builtin does nothing for a non-caching EDD application because it does not have a cache. Updates the device with all of the values in the cache that the method has changed.

Lexical structure

send_all_values(VOID)

The lexical element of the Builtin send_all_values is shown in Table 291.

Table 291 – Builtin send_all_values

Parameter name	Type	Direction	Usage	Description
<return>	long	O	m	One of the return codes specified in Table 366.

4.288 Builtin send_command

Purpose

The Builtin send_command will send the specified command. If the status bit indicating that more data is available is returned (status class MORE), command 48 is automatically issued. The status/data octets from the original command and command 48 (if it was sent) are then logically ANDed with the appropriate abort masks to determine if the method should be aborted. If there is a match, the method is aborted and the abort methods (if any were designated) are run. If there is no match for the abort masks, the same data is ANDed with the appropriate retry masks. If there is a match, the appropriate command will be retried (either the specified command, or command 48). This continues until either the command

succeeds, an abort occurs, or the maximum number of retries is reached. No status or data octets are returned with this function.

Lexical structure

`send_command(cmd_number)`

The lexical elements of the Builtin `send_command` are shown in Table 292.

Table 292 – Builtin `send_command`

Parameter name	Type	Direction	Usage	Description
<code>cmd_number</code>	int	I	m	The number of command to send.
<return>	int	O	m	One of the return codes specified in Table 367.

4.289 Builtin `send_command_trans`

Purpose

The Builtin `send_command_trans` sends the command with the specified transaction to the device. This function is to be used to send commands that have been defined with multiple transactions.

The Builtin `send_command_trans` will send the specified command. If the status bit indicating that more data is available is returned (status class MORE), command 48 is automatically issued. The status/data octets from the original command and command 48 (if it was sent) are then logically ANDed with the appropriate abort masks to determine if the method should be aborted. If there is a match, the method aborts. The same data is ANDed with the appropriate retry masks. If there is a match, the appropriate command will be retried (either the specified command, or command 48). This continues until either the command succeeds, an abort occurs, or the maximum number of retries is reached. No status or data octets are returned with this command.

Lexical structure

`send_command_trans(cmd_number, transaction)`

The lexical elements of the Builtin `send_command_trans` are shown in Table 293.

Table 293 – Builtin `send_command_trans`

Parameter name	Type	Direction	Usage	Description
<code>cmd_number</code>	int	I	m	The number of command to send.
<code>transaction</code>	int	I	m	The number of the TRANSACTION.
<return>	int	O	m	One of the return codes specified in Table 367.

4.290 Builtin `send_on_exit`

Purpose

The Builtin `send_on_exit` sets the termination action to send any changes made by the method to variables stored in the cache of the EDD application, which have not already been sent to the device. This Builtin does nothing in a non-caching EDD application, because a non-caching EDD application does not have a cache.

When a method modifies a variable value, it changes a copy of that value. This change is effective only for the life of the method. In order to make the change in the device, the method shall send the value(s) to the device or call this Builtin before the method exits.

All values that are changed, but not sent to the device, are handled at the termination of the method by either sending them to the device, discarding them, or saving the value changes in a way that survives the end of the method. The changed values in the cache are sent after the method terminates.

If a method does not call Builtin discard_on_exit, or Builtin send_on_exit prior to the exiting, any values changed by the Builtin are discarded as if Builtin discard_on_exit was called. If the method aborts then the Builtin discard_on_exit option is enforced, even if Builtin save_on_exit, or Builtin send_on_exit were called before the method aborted.

Lexical structure

send_on_exit(VOID)

The lexical element of the Builtin send_on_exit is shown in Table 294.

Table 294 – Builtin send_on_exit

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.291 Builtin send_trans

Purpose

The Builtin send_trans sends a command with the specified transaction to the device. This function is to be used to send a command that has been defined with multiple transactions.

The Builtin send_trans will send the specified command and return the response code, communications status, and command status respectively in the cmd_status array. These octets are logically ANDed with the appropriate abort masks to determine if the method should be aborted. If there is a match, the method is aborted and the abort methods (if any were designated) are run. If there is a match, the command will be retried. This continues until either the command succeeds, an abort occurs, or the maximum number of retries is reached.

The status octets for the command are returned in the cmd_status parameter, a 3-octet array, allocated in the method making the call.

Lexical structure

send_trans(cmd_number, transaction, cmd_status)

The lexical elements of the Builtin send_trans are shown in Table 295.

Table 295 – Builtin send_trans

Parameter name	Type	Direction	Usage	Description
cmd_number	int	I	m	The number of command to send.
transaction	int	I	m	The number of the TRANSACTION.
cmd_status	char*	O	m	The status octets of the send command.
<return>	int	O	m	One of the return codes specified in Table 367.

4.292 Builtin send_value

Purpose

The Builtin send_value causes the value of the specked variable in the cache to be sent to the device, thus updating the device value. This Builtin does nothing in a non-caching EDD

application, because a non-caching EDD application does not have a cache. Some values are READ_ONLY in the device and cannot be written. Attempting to do so will cause a Builtin error to be returned.

The parameters ID and member ID are specified using the Builtins ITEM_ID and MEMBER_ID and the name or reference of the variable in question.

Lexical structure

```
send_value(id, member_id)
```

The lexical elements of the Builtin send_value are shown in Table 296.

Table 296 – Builtin send_value

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	m	The item ID of the VARIABLE to read.
member_id	unsigned long	I	m	The member ID of the VARIABLE to read.
<return>	long	O	m	One of the return codes specified in Table 366.

4.293 Builtin send_value2

Purpose

The Builtin send_value2 causes the value of the specked variable in the cache to be sent to the device, thus updating the device value. This Builtin does nothing in a non-caching EDD application, because a non-caching EDD application does not have a cache. Some values are READ_ONLY in the device and cannot be written. Attempting to do so will cause a Builtin error to be returned.

The parameters ID and member ID are specified using the Builtins ITEM_ID and MEMBER_ID and the name or reference of the variable in question.

Lexical structure

```
send_value2(blk_id, blk_num, id, member_id)
```

The lexical elements of the Builtin send_value2 are shown in Table 297.

Table 297 – Builtin send_value2

Parameter name	Type	Direction	Usage	Description
blk_id	unsigned long	I	m	The item ID of the BLOCK_A instance to access.
blk_num	unsigned long	I	m	The occurrence number of the BLOCK_A instance to access.
id	unsigned long	I	m	The item ID of the VARIABLE to read.
member_id	unsigned long	I	m	The member ID of the VARIABLE to read.
<return>	long	O	m	One of the return codes specified in Table 366.

4.294 Builtin SET_NUMBER_OF_RETRIES

Purpose

The Builtin SET_NUMBER_OF_RETRIES sets up the number of times a command will be retried due to the appropriate retry masks. This value is defaulted to 0 at the beginning of each method.

Lexical structure

SET_NUMBER_OF_RETRIES (n)

The lexical elements of the Builtin SET_NUMBER_OF_RETRIES are shown in Table 298.

Table 298 – Builtin SET_NUMBER_OF_RETRIES

Parameter name	Type	Direction	Usage	Description
n	int	I	m	The number of retries of a command
<return>	int	O	m	One of the return codes specified in Table 367.

4.295 Builtin sgetval

Purpose

The Builtin sgetval gets the value of a string type (ASCII, PACKED_ASCII, PASSWORD, EUC) VARIABLE. It gets the value of the variable from which the method is called as an action (e.g. PRE_EDIT_ACTIONS).

Lexical structure

sgetval (value)

The lexical elements of the Builtin sgetval are shown in Table 299.

Table 299 – Builtin sgetval

Parameter name	Type	Direction	Usage	Description
value	DD_STRING	O	m	The value of the VARIABLE.
<return>	int	O	m	One of the return codes specified in Table 367.

4.296 Builtin ShortToByte

Purpose

The Builtin ShortToByte will provide a vector of unsigned char values representing the particular elements of a short value.

Lexical structure

ShortToByte(in, out1, out2)

The lexical elements of the Builtin ShortToByte are shown in Table 300.

Table 300 – Builtin ShortToByte

Parameter name	Type	Direction	Usage	Description
in	short	I	m	The short value to convert.
out1	unsigned char	O	m	First element of the short value (most significant byte).
out2	unsigned char	O	m	Second element of the short value (least significant byte).
<return>	VOID	—	—	—

4.297 Builtin sin**Purpose**

The Builtin sin returns the sine a floating-point value.

Lexical structure

`sin(x)`

The lexical elements of the Builtin sin are shown in Table 301.

Table 301 – Builtin sin

Parameter name	Type	Direction	Usage	Description
x	double	I	m	A floating-point value.
<return>	double	O	m	The sine.

4.298 Builtin sinh**Purpose**

The Builtin sinh returns the hyperbolic sine of a floating-point value.

Lexical structure

`sinh(x)`

The lexical elements of the Builtin sinh are shown in Table 302.

Table 302 – Builtin sinh

Parameter name	Type	Direction	Usage	Description
x	double	I	m	A floating-point value.
<return>	double	O	m	The hyperbolic sine.

4.299 Builtin sqrt**Purpose**

The Builtin sqrt returns the square root of a floating-point value.

Lexical structure

`sqrt(x)`

The lexical elements of the Builtin sqrt are shown in Table 303.

Table 303 – Builtin sqrt

Parameter name	Type	Direction	Usage	Description
x	double	I	m	A floating-point value.
<return>	double	O	m	The square root.

4.300 Builtin ssetval

Purpose

The Builtin ssetval sets the value of a string type (ASCII, PACKED_ASCII, PASSWORD, EUC) VARIABLE. It sets the value of the variable from which the method was called as an action (e.g. PRE_EDIT_ACTIONS).

Lexical structure

ssetval(value)

The lexical elements of the Builtin ssetval are shown in Table 304.

Table 304 – Builtin ssetval

Parameter name	Type	Direction	Usage	Description
value	DD_STRING	I	m	The value of the VARIABLE.
<return>	DD_STRING	O	m	The value of the VARIABLE.

4.301 Builtin strcmp

Purpose

The Builtin strcmp compares two strings.

Lexical structure

strcmp(string1, string2)

The lexical elements of the Builtin strcmp are shown in Table 305.

Table 305 – Builtin strcmp

Parameter name	Type	Direction	Usage	Description
string1	DD_STRING	I	m	The first string.
string2	DD_STRING	I	m	The second string.
<return>	int	O	m	Zero if string1 and string2 are the same, returns a negative value if string1 is less than string2, returns a positive value if string1 is greater than string2.

4.302 Builtin strleft

Purpose

The Builtin strleft extracts letters from the beginning of a string.

Lexical structure

strleft(string, length)

The lexical elements of the Builtin strleft are shown in Table 306.

Table 306 – Builtin strleft

Parameter name	Type	Direction	Usage	Description
string	DD_STRING	I	m	The input string.
length	int	I	m	The number of characters to be extracted from the input string.
<return>	DD_STRING	O	m	The specified substring.

4.303 Builtin strlen**Purpose**

The Builtin strlen returns the length of a string.

Lexical structure

strlen(string)

The lexical elements of the Builtin strlen are shown in Table 307.

Table 307 – Builtin strlen

Parameter name	Type	Direction	Usage	Description
string	DD_STRING	I	m	The string whose length is to be returned.
<return>	int	O	m	The length of the string.

4.304 Builtin strlwr**Purpose**

The Builtin strlwr returns a lower-case version of a string.

Lexical structure

strlwr(string)

The lexical elements of the Builtin strlwr are shown in Table 308.

Table 308 – Builtin strlwr

Parameter name	Type	Direction	Usage	Description
string	DD_STRING	I	m	The string to be converted.
<return>	DD_STRING	O	m	A lower-case version of the string.

4.305 Builtin strmid**Purpose**

The Builtin strmid returns a substring of a string.

Lexical structure

strmid(string, position, length)

The lexical elements of the Builtin strmid are shown in Table 309.

Table 309 – Builtin strmid

Parameter name	Type	Direction	Usage	Description
string	DD_STRING	I	m	The input string.
position	int	I	m	The first character to be extracted from the input string.
length	int	I	m	The number of characters to be extracted from the input string.
<return>	DD_STRING	O	m	The specified substring.

4.306 Builtin strright

Purpose

The Builtin strright extracts letters from the end of a string.

Lexical structure

strright(string, length)

The lexical elements of the Builtin strright are shown in Table 310.

Table 310 – Builtin strright

Parameter name	Type	Direction	Usage	Description
string	DD_STRING	I	m	The input string.
length	int	I	m	The number of characters to be extracted from the input string.
<return>	DD_STRING	O	m	The specified substring.

4.307 Builtin strstr

Purpose

The Builtin strstr returns a substring of a string.

Lexical structure

strstr(string, substring)

The lexical elements of the Builtin strstr are shown in Table 311.

Table 311 – Builtin strstr

Parameter name	Type	Direction	Usage	Description
string	DD_STRING	I	m	The string to be searched.
substring	DD_STRING			The search string.
<return>	DD_STRING	O	m	The first occurrence of s1 in s2, or NULL if the string cannot be found.

4.308 Builtin strtrim

Purpose

The Builtin strtrim trims whitespace from a string.

Lexical structure

```
strtrim(string)
```

The lexical elements of the Builtin strtrim are shown in Table 312.

Table 312 – Builtin strtrim

Parameter name	Type	Direction	Usage	Description
string	DD_STRING	I	m	The string to be trimmed.
<return>	DD_STRING	O	m	A string with leading and trailing whitespace removed.

4.309 Builtinstrupr**Purpose**

The Builtinstrupr returns an upper-case version of a string.

Lexical structure

```
strupr(string)
```

The lexical elements of the Builtinstrupr are shown in Table 313.

Table 313 – Builtinstrupr

Parameter name	Type	Direction	Usage	Description
string	DD_STRING	I	m	The string to be converted.
<return>	DD_STRING	O	m	An upper-case version of the string.

4.310 Builtin tan**Purpose**

The Builtin tan returns the tangent of a floating-point value.

Lexical structure

```
tan(x)
```

The lexical elements of the Builtin tan are shown in Table 314.

Table 314 – Builtin tan

Parameter name	Type	Direction	Usage	Description
x	double	I	m	A floating-point value.
<return>	double	O	m	The tangent.

4.311 Builtin tanh**Purpose**

The Builtin tanh returns the hyperbolic tangent of a floating-point value.

Lexical structure

```
tanh(x)
```


The lexical elements of the Builtin tanh are shown in Table 315.

Table 315 – Builtin tanh

Parameter name	Type	Direction	Usage	Description
x	double	I	m	A floating-point value.
<return>	double	O	m	The hyperbolic tangent.

4.312 Builtin Time_To_Date

Purpose

The Builtin Time_To_Date creates a DATE value from a calendar date and time.

Lexical structure

Time_To_Date(t)

The lexical elements of the Builtin Time_To_Date are shown in Table 316.

Table 316 – Builtin Time_To_Date

Parameter name	Type	Direction	Usage	Description
t	time_t	I	m	The time_t value.
<return>	long	O	m	The DATE value.

4.313 Builtin TIME_VALUE_to_Hour

Purpose

The Builtin TIME_VALUE_to_Hour extracts the hour component from a TIME_VALUE.

Lexical structure

TIME_VALUE_to_Hour(time_value)

The lexical elements of the Builtin TIME_VALUE_to_Hour are shown in Table 317.

Table 317 – Builtin TIME_VALUE_to_Hour

Parameter name	Type	Direction	Usage	Description
time_value	unsigned long or unsigned long long	I	m	The TIME_VALUE. A TIME_VALUE(8) is passed in as an unsigned long long and is an absolute time. A TIME_VALUE(4) is passed in as an unsigned long and is a duration or a time of day.
<return>	int	O	m	The hour component.

4.314 Builtin TIME_VALUE_to_Minute

Purpose

The Builtin TIME_VALUE_to_Minute extracts the minute component from a TIME_VALUE.

Lexical structure

TIME_VALUE_to_Minute(time_value)

The lexical elements of the Builtin TIME_VALUE_to_Minute are shown in Table 318.

Table 318 – Builtin TIME_VALUE_to_Minute

Parameter name	Type	Direction	Usage	Description
time_value	unsigned long or unsigned long long	I	m	The TIME_VALUE. A TIME_VALUE(8) is passed in as an unsigned long long and is an absolute time. A TIME_VALUE(4) is passed in as an unsigned long and is a duration or a time of day.
<return>	int	O	m	The minute component.

4.315 Builtin TIME_VALUE_to_Second

Purpose

The Builtin TIME_VALUE_to_Second extracts the seconds component from a TIME_VALUE.

Lexical structure

TIME_VALUE_to_Second(time_value)

The lexical elements of the Builtin TIME_VALUE_to_Second are shown in Table 319.

Table 319 – Builtin TIME_VALUE_to_Second

Parameter name	Type	Direction	Usage	Description
time_value	unsigned long or unsigned long long	I	M	The TIME_VALUE. A TIME_VALUE(8) is passed in as an unsigned long long and is an absolute time. A TIME_VALUE(4) is passed in as an unsigned long and is a duration or a time of day.
<return>	int	O	m	The seconds component.

4.316 Builtin TIME_VALUE_to_seconds

Purpose

The Builtin TIME_VALUE_to_seconds converts a TIME_VALUE to a number of seconds.

Lexical structure

TIME_VALUE_to_seconds(time_value)

The lexical elements of the Builtin TIME_VALUE_to_seconds are shown in Table 320.

Table 320 – Builtin TIME_VALUE_to_seconds

Parameter name	Type	Direction	Usage	Description
time_value	unsigned long or unsigned long long	I	m	The TIME_VALUE. A TIME_VALUE(8) is passed in as an unsigned long long and is an absolute time. A TIME_VALUE(4) is passed in as an unsigned long and is a duration or a time of day.
<return>	double	O	m	The number of seconds of the duration or seconds since midnight if a TIME_VALUE(4) was given or the number of seconds since January 1 st 1972 if a TIME_VALUE(8) was given.

4.317 Builtin TIME_VALUE_to_string

Purpose

The Builtin TIME_VALUE_to_string creates a string representation of a TIME_VALUE. The programming language-C strftime function defines the structure of the format string.

Lexical structure

TIME_VALUE_to_string(string, format, time_value)

The lexical elements of the Builtin TIME_VALUE_to_string are shown in Table 321.

Table 321 – Builtin TIME_VALUE_to_string

Parameter name	Type	Direction	Usage	Description
string	DD_STRING	O	m	The string into which the formatted TIME_VALUE is copied.
format	DD_STRING	I	m	How the TIME_VALUE is to be formatted.
time_value	unsigned long or unsigned long long	I	m	The TIME_VALUE to be formatted. A TIME_VALUE(8) is passed in as an unsigned long long and is an absolute time. A TIME_VALUE(4) is passed in as an unsigned long and is a duration or a time of day.
<return>	int	O	m	The number of characters actually copied or -1 if an error occurs.

4.318 Builtin timet_to_string

Purpose

The Builtin timet_to_string creates a string representation of a time_t. The programming language-C strftime function defines the structure of the format string.

Lexical structure

timet_to_string(string, format, timet)

The lexical elements of the Builtin timet_to_string are shown in Table 322.

Table 322 – Builtin `timet_to_string`

Parameter name	Type	Direction	Usage	Description
string	DD_STRING	O	m	The string into which formatted <code>time_t</code> is copied.
format	DD_STRING	I	m	How the <code>time_t</code> is to be formatted.
timet	<code>time_t</code>	I	m	The <code>time_t</code> to be formatted.
<return>	int	O	m	The number of characters actually copied or -1 is an error occurs.

4.319 Builtin `timet_to_TIME_VALUE`**Purpose**

The Builtin `timet_to_TIME_VALUE` converts the time of day part of a `time_t` to a `TIME_VALUE(4)`.

Lexical structure

```
timet_to_TIME_VALUE(timet)
```

The lexical elements of the Builtin `timet_to_TIME_VALUE` are shown in Table 323.

Table 323 – Builtin `timet_to_TIME_VALUE`

Parameter name	Type	Direction	Usage	Description
timet	<code>time_t</code>	I	m	The <code>time_t</code> value to convert.
<return>	unsigned long	O	m	The <code>TIME_VALUE(4)</code> value corresponding to the time of day given by the input parameter <code>timet</code> .

4.320 Builtin `timet_to_TIME_VALUE8`**Purpose**

The Builtin `timet_to_TIME_VALUE8` converts `time_t` to a `TIME_VALUE(8)`.

Lexical structure

```
timet_to_TIME_VALUE8(timet)
```

The lexical elements of the Builtin `timet_to_TIME_VALUE8` are shown in Table 324.

Table 324 – Builtin `timet_to_TIME_VALUE8`

Parameter name	Type	Direction	Usage	Description
timet	<code>time_t</code>	I	m	The <code>time_t</code> value to convert.
<return>	unsigned long long	O	m	The <code>TIME_VALUE(8)</code> value corresponding to the date and time given by the input parameter <code>timet</code> .

4.321 Builtin `To_Date`**Purpose**

The Builtin `To_Date` creates a `DATE` from year, month and day.

Lexical structure

```
To_Date(year, month, day)
```

The lexical elements of the Builtin To_Date are shown in Table 325.

Table 325 – Builtin To_Date

Parameter name	Type	Direction	Usage	Description
year	int	I	m	The year.
month	int	I	m	The month.
day	int	I	m	The day.
<return>	long	O	m	The DATE value.

4.322 Builtin To_Date_and_Time

Purpose

The Builtin To_Date_and_Time converts a days, hour, minute, second and millisecond value to a DATE_AND_TIME value.

NOTE The return value can be assigned to a DATE_AND_TIME variable.

Lexical structure

To_Date_and_Time(days, hour, minute, second, millisecond)

The lexical elements of the Builtin To_Date_and_Time are shown in Table 326.

Table 326 – Builtin To_Date_and_Time

Parameter name	Type	Direction	Usage	Description
days	int	I	m	The number of days since January 1 st , 1900. The number of days can only be a positive number.
hour	int	I	m	The hour
minute	int	I	m	The minute
second	int	I	m	The second
millisecond	int	I	m	The millisecond
<return>	unsigned long long	O	m	The DATE_AND_TIME value corresponding to the given time_t value packaged in an unsigned long long value.

4.323 Builtin To_Time

Purpose

The Builtin To_Time creates a time_t value from its input arguments.

Lexical structure

To_Time(date, hour, minute, second, isDST)

The lexical elements of the Builtin To_Time are shown in Table 327.

Table 327 – Builtin To_Time

Parameter name	Type	Direction	Usage	Description
date	long	I	m	A long value containing the DATE.
hour	int	I	m	The hour.
minute	int	I	m	The minute.
second	int	I	m	The second.
isDST	int	I	m	Specifies whether daylight savings time is in effect. If zero, daylight savings time is not in effect; otherwise, daylight savings time is in effect.
<return>	time_t	O	m	The time_t value.

4.324 Builtin To_TIME_VALUE**Purpose**

The Builtin To_TIME_VALUE creates a TIME_VALUE from its input arguments.

Lexical structure

To_TIME_VALUE(hours, minutes, seconds)

The lexical elements of the Builtin To_TIME_VALUE are shown in Table 328.

Table 328 – Builtin To_TIME_VALUE

Parameter name	Type	Direction	Usage	Description
hours	int	I	m	The number of hours.
minutes	int	I	m	The number of minutes.
seconds	int	I	m	The number of seconds.
<return>	unsigned long	O	m	The TIME_VALUE(4) value corresponding to the time of day given by the input parameter hours, minutes and seconds.

4.325 Builtin To_TIME_VALUE8**Purpose**

The Builtin To_TIME_VALUE8 creates a TIME_VALUE(8) from its input arguments.

Lexical structure

To_TIME_VALUE8(year, month, day, hour, minute, second)

The lexical elements of the Builtin To_TIME_VALUE are shown in Table 329.

Table 329 – Builtin To_TIME_VALUE8

Parameter name	Type	Direction	Usage	Description
year	int	I	m	The year.
month	int	I	m	The month.
day	int	I	m	The day.
hour	int	I	m	The hour.
minute	int	I	m	The minute.
second	int	I	m	The second.
<return>	unsigned long long	O	m	The TIME_VALUE(8) value corresponding to the input parameters.

4.326 Builtin trunc**Purpose**

The Builtin trunc truncates a floating-point value.

Lexical structure

`trunc(x)`

The lexical elements of the Builtin trunc are shown in Table 330.

Table 330 – Builtin trunc

Parameter name	Type	Direction	Usage	Description
x	double	I	m	A floating-point value.
<return>	double	O	m	The truncated value.

4.327 Builtin VARID**Purpose**

The Builtin VARID will return the numeric identifier for the variable specified. A valid variable name shall be provided. Each variable in the device description is assigned a unique numeric identifier. This function is to be used when the identifier of a variable either needs to be stored in a temporary buffer, or needs to be sent as a parameter to another Builtin.

Lexical structure

`VARID(variable_name)`

The lexical elements of the Builtin VARID are shown in Table 331

Table 331 – Builtin VARID

Parameter name	Type	Direction	Usage	Description
variable_name	reference	I	m	The identifier of the VARIABLE.
<return>	int	O	m	The variable identifier.

4.328 Builtin vassign

Purpose

The Builtin vassign will assign the value of the source variable to the destination variable. Both variables shall be valid and they shall have the same type.

Lexical structure

```
vassign(dest_var, source_var)
```

The lexical elements of the Builtin vassign are shown in Table 332.

Table 332 – Builtin vassign

Parameter name	Type	Direction	Usage	Description
dest_var	reference	I	m	The identifier of the destination VARIABLE.
source_var	reference	I	m	The identifier of the source VARIABLE.
<return>	int	O	m	One of the return codes specified in Table 367.

4.329 Builtin WriteCommand

Purpose

The Builtin WriteCommand allows to write variables with a given command.

Lexical structure

```
WriteCommand(command_identifier)
```

The lexical elements of the Builtin WriteCommand are shown in Table 333.

Table 333 – Builtin WriteCommand

Parameter name	Type	Direction	Usage	Description
command_identifier	reference	I	m	The Identifier of the COMMAND to execute.
<return>	long	O	m	A protocol specific error value, the encoding of which is defined by the specific protocol or device.

4.330 Builtin writeItemToDevice

Purpose

The Builtin writeItemToDevice writes an item from an open block transfer indicated by the handle. The EDD_ITEM may be a compound item but shall contain only data.

Lexical structure

```
writeItemToDevice(handle, dd_item)
```

The lexical elements of the Builtin writeItemToDevice are shown in Table 334.

Table 334 – Builtin writeltemToDevice

Parameter name	Type	Direction	Usage	Description
handle	unsigned int	I	m	The handle of the block transfer.
dd_item	DD_ITEM &	O	m	The DD item to write.
<return>	int	O	m	One of the return codes specified in Table 367.

4.331 Builtin XMTR_ABORT_ON_ALL_COMM_STATUS**Purpose**

The Builtin XMTR_ABORT_ON_ALL_COMM_STATUS will set all of the bits in the command 48 comm status abort mask. This will cause the system to abort the current method if the device returns any command 48 comm status value. Comm status is defined to be the first octet returned in a transaction, when bit 7 of this octet is set.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_ABORT_ON_ALL_COMM_STATUS(VOID)

The lexical element of the Builtin XMTR_ABORT_ON_ALL_COMM_STATUS is shown in Table 335.

Table 335 – Builtin XMTR_ABORT_ON_ALL_COMM_STATUS

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.332 Builtin XMTR_ABORT_ON_ALL_DATA**Purpose**

The Builtin XMTR_ABORT_ON_ALL_DATA will set all of the bits in the command 48 abort masks. This will cause the system to abort if any bit in command 48 is set.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will be valid only during the current method. See send_command for implementation of the masks. See ABORT_ON_RESPONSE_CODE for a list of the available masks, and their default values.

The mask affected by this function is used by the send, send_trans, ext_send, ext_send_trans, get_more_status, and display functions.

Lexical structure

XMTR_ABORT_ON_ALL_DATA(VOID)

The lexical element of the Builtin XMTR_ABORT_ON_ALL_DATA is shown in Table 336.

Table 336 – Builtin XMTR_ABORT_ON_ALL_DATA

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.333 Builtin XMTR_ABORT_ON_ALL_DEVICE_STATUS**Purpose**

The Builtin XMTR_ABORT_ON_ALL_DEVICE_STATUS will set all of the bits in the command 48 device status abort mask. This will cause the system to abort the current method if the device returns any command 48 device status value. Device status is defined to be the second data octet returned in a transaction.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_ABORT_ON_ALL_DEVICE_STATUS (VOID)

The lexical element of the Builtin XMTR_ABORT_ON_ALL_DEVICE_STATUS is shown in Table 337.

Table 337 – Builtin XMTR_ABORT_ON_ALL_DEVICE_STATUS

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.334 Builtin XMTR_ABORT_ON_ALL_RESPONSE_CODES**Purpose**

The Builtin XMTR_ABORT_ON_ALL_RESPONSE_CODES will set all of the bits in the command 48 response code abort mask. This will cause the system to abort the current method if the device returns any command 48 response code value.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_ABORT_ON_ALL_RESPONSE_CODES (VOID)

The lexical element of the Builtin XMTR_ABORT_ON_ALL_RESPONSE_CODES is shown in Table 338.

Table 338 – Builtin XMTR_ABORT_ON_ALL_RESPONSE_CODES

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.335 Builtin XMTR_ABORT_ON_COMM_ERROR**Purpose**

The Builtin XMTR_ABORT_ON_COMM_ERROR will set the command 48 no comm error mask such that the method will be aborted if a comm error is found while sending command 48.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_ABORT_ON_COMM_ERROR(VOID)

The lexical element of the Builtin XMTR_ABORT_ON_COMM_ERROR is shown in Table 339.

Table 339 – Builtin XMTR_ABORT_ON_COMM_ERROR

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.336 Builtin XMTR_ABORT_ON_COMM_STATUS**Purpose**

The Builtin XMTR_ABORT_ON_COMM_STATUS will set the correct bit(s) in the command 48 comm status abort mask such that the specified command 48 comm status value will cause the method to abort. Comm status is defined to be the first data octet returned in a transaction, when bit 7 of this octet is 1. The comm_status_value can be set to a value from 0 to 127, where each bit corresponds to a communications error received from the device.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_ABORT_ON_COMM_STATUS(comm_status)

The lexical elements of the Builtin XMTR_ABORT_ON_COMM_STATUS are shown in Table 340.

Table 340 – Builtin XMTR_ABORT_ON_COMM_STATUS

Parameter name	Type	Direction	Usage	Description
comm_status	int	l	m	The new comm status abort mask.
<return>	VOID	—	—	—

4.337 Builtin XMTR_ABORT_ON_DATA**Purpose**

The Builtin XMTR_ABORT_ON_DATA will set the correct bit(s) in the abort data mask such that the method will be aborted if the specified bit is set in the data field returned by command 48.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_ABORT_ON_DATA(data, bit)

The lexical elements of the Builtin XMTR_ABORT_ON_DATA are shown in Table 341.

Table 341 – Builtin XMTR_ABORT_ON_DATA

Parameter name	Type	Direction	Usage	Description
data	int	l	m	Zero based index into command 48 data field.
bit	int	l	m	Bit number (0..7). Bit 0 is the least significant bit.
<return>	VOID	—	—	—

4.338 Builtin XMTR_ABORT_ON_DEVICE_STATUS**Purpose**

The Builtin XMTR_ABORT_ON_DEVICE_STATUS will set the correct bit(s) in the command 48 device status abort mask in such a way that the specified command 48 device status value will cause the method to abort. Device status is defined to be the second data octet returned in a transaction. The device_status value can be set to a value from 0 to 127, where each bit corresponds to a device status returned from the device.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

Lexical structure

XMTR_ABORT_ON_DEVICE_STATUS(device_status)

The lexical elements of the Builtin XMTR_ABORT_ON_DEVICE_STATUS are shown in Table 342.

Table 342 – Builtin XMTR_ABORT_ON_DEVICE_STATUS

Parameter name	Type	Direction	Usage	Description
device_status	int	I	m	The new device status abort mask.
<return>	VOID	—	—	—

4.339 Builtin XMTR_ABORT_ON_NO_DEVICE

Purpose

The Builtin XMTR_ABORT_ON_NO_DEVICE will set the command 48 no device mask such that the method will be aborted if no device is found while sending a command 48.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_ABORT_ON_NO_DEVICE (VOID)

The lexical element of the Builtin XMTR_ABORT_ON_NO_DEVICE is shown in Table 343.

Table 343 – Builtin XMTR_ABORT_ON_NO_DEVICE

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.340 Builtin XMTR_ABORT_ON_RESPONSE_CODE

Purpose

The Builtin XMTR_ABORT_ON_RESPONSE_CODE will set the correct bit(s) in the response code abort mask such that the specified command 48 response code value will cause the method to abort. The response code is defined to be the first data octet returned in a transaction, when bit 7 of this octet is 0.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_ABORT_ON_RESPONSE_CODE (response_code)

The lexical elements of the Builtin XMTR_ABORT_ON_RESPONSE_CODE are shown in Table 344.

Table 344 – Builtin XMTR_ABORT_ON_RESPONSE_CODE

Parameter name	Type	Direction	Usage	Description
response_code	int	l	m	The new response code abort mask.
<return>	VOID	—	—	—

4.341 Builtin XMTR_IGNORE_ALL_COMM_STATUS**Purpose**

The Builtin XMTR_IGNORE_ALL_COMM_STATUS will clear all of the bits in the command 48 comm status retry and abort masks. This will cause the system to ignore all bits in the comm status value. Comm status is defined to be the first data octet returned in a transaction, when bit 7 of this octet is set.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_IGNORE_ALL_COMM_STATUS (VOID)

The lexical element of the Builtin XMTR_IGNORE_ALL_COMM_STATUS is shown in Table 345.

Table 345 – Builtin XMTR_IGNORE_ALL_COMM_STATUS

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.342 Builtin XMTR_IGNORE_ALL_DATA**Purpose**

The Builtin XMTR_IGNORE_ALL_DATA will clear all of the bits in the command 48 device status retry and abort masks. This will cause the system to ignore all data in the command 48 response message.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will be valid only during the current method. See send_command for implementation of the masks. See ABORT_ON_RESPONSE_CODE for a list of the available masks, and their default values.

The mask affected by this function is used by the send, send_trans, ext_send, ext_send_trans, get_more_status, and display functions.

Lexical structure

XMTR_IGNORE_ALL_DATA (VOID)

The lexical element of the Builtin XMTR_IGNORE_ALL_DATA is shown in Table 346.

Table 346 – Builtin XMTR_IGNORE_ALL_DATA

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.343 Builtin XMTR_IGNORE_ALL_DEVICE_STATUS**Purpose**

The Builtin XMTR_IGNORE_ALL_DEVICE_STATUS will clear all of the bits in the command 48 device status retry and abort masks. This will cause the system to ignore all bits in the command 48 device status octet. Device status is defined to be the second data octet returned in a transaction.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_IGNORE_ALL_DEVICE_STATUS (VOID)

The lexical element of the Builtin XMTR_IGNORE_ALL_DEVICE_STATUS is shown in Table 347.

Table 347 – Builtin XMTR_IGNORE_ALL_DEVICE_STATUS

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.344 Builtin XMTR_IGNORE_ALL_RESPONSE_CODES**Purpose**

The Builtin XMTR_IGNORE_ALL_RESPONSE_CODES will clear all of the bits in the command 48 response code retry and abort masks. This will cause the system to ignore all response code values returned from the device.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_IGNORE_ALL_RESPONSE_CODES (VOID)

The lexical element of the Builtin XMTR_IGNORE_ALL_RESPONSE_CODES is shown in Table 348.

Table 348 – Builtin XMTR_IGNORE_ALL_RESPONSE_CODES

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.345 Builtin XMTR_IGNORE_COMM_ERROR**Purpose**

The Builtin XMTR_IGNORE_COMM_ERROR will set the command 48 comm error mask such that a communications error condition will be ignored while sending command 48.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_IGNORE_COMM_ERROR(VOID)

The lexical element of the Builtin XMTR_IGNORE_COMM_ERROR is shown in Table 349.

Table 349 – Builtin XMTR_IGNORE_COMM_ERROR

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.346 Builtin XMTR_IGNORE_COMM_STATUS**Purpose**

The Builtin XMTR_IGNORE_COMM_STATUS will clear the correct bit(s) in the command 48 comm status abort and retry mask such that the specified bits in the command 48 comm status value will be ignored. Comm status is defined to be the first data octet returned in a transaction, when bit 7 of this octet is set.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_IGNORE_COMM_STATUS(comm_status)

The lexical elements of the Builtin XMTR_IGNORE_COMM_STATUS are shown in Table 350.

Table 350 – Builtin XMTR_IGNORE_COMM_STATUS

Parameter name	Type	Direction	Usage	Description
comm_status	int	l	m	The new comm status abort and retry mask.
<return>	VOID	—	—	—

4.347 Builtin XMTR_IGNORE_DATA

Purpose

The Builtin XMTR_IGNORE_DATA shall reset the specified bit, in the specified byte, in the abort data mask such that the method will ignore the specified bit in the data field returned by command 48.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will be valid only during the current method. See send_command for implementation of the masks.

See ABORT_ON_RESPONSE_CODE for a list of the available masks, and their default values.

The mask affected by this function is used by the send, send_trans, ext_send, ext_send_trans, get_more_status, and display functions.

Lexical structure

XMTR_IGNORE_DATA(data, bit)

The lexical elements of the Builtin XMTR_IGNORE_DATA are shown in Table 351.

Table 351 – Builtin XMTR_IGNORE_DATA

Parameter name	Type	Direction	Usage	Description
Data	int	l	m	Zero based index into command 48 data field.
Bit	int	l	m	Bit number (0..7). Bit 0 is the least significant bit.
<return>	VOID	—	—	—

4.348 Builtin XMTR_IGNORE_DEVICE_STATUS

Purpose

The Builtin XMTR_IGNORE_DEVICE_STATUS will clear the correct bit(s) in the command 48 device status abort and retry masks such that the specified bits in the command 48 device status octet are ignored. Device status is defined to be the second data octet returned in a transaction.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_IGNORE_DEVICE_STATUS(device_status)

The lexical elements of the Builtin XMTR_IGNORE_DEVICE_STATUS are shown in Table 352.

Table 352 – Builtin XMTR_IGNORE_DEVICE_STATUS

Parameter name	Type	Direction	Usage	Description
device_status	int	l	m	The new device status abort and retry mask.
<return>	VOID	—	—	—

4.349 Builtin XMTR_IGNORE_NO_DEVICE

Purpose

The Builtin XMTR_IGNORE_NO_DEVICE will set the command 48 no device mask to show that the no device condition should be ignored while sending a command 48.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_IGNORE_NO_DEVICE(VOID)

The lexical element of the Builtin XMTR_IGNORE_NO_DEVICE is shown in Table 353.

Table 353 – Builtin XMTR_IGNORE_NO_DEVICE

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.350 Builtin XMTR_IGNORE_RESPONSE_CODE

Purpose

The Builtin XMTR_IGNORE_RESPONSE_CODE will clear the correct bit(s) in the command 48 response code masks in such a way that the specified command 48 response code value will be ignored. The response code is defined to be the first data octet returned in a transaction, when bit 7 of this octet is 0.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_IGNORE_RESPONSE_CODE(response_code)

The lexical elements of the Builtin XMTR_IGNORE_RESPONSE_CODE are shown in Table 354.

Table 354 – Builtin XMTR_IGNORE_RESPONSE_CODE

Parameter name	Type	Direction	Usage	Description
response_code	int	l	m	The new response code mask.
<return>	VOID	—	—	—

4.351 Builtin XMTR_RETRY_ON_ALL_COMM_STATUS

Purpose

The Builtin XMTR_RETRY_ON_ALL_COMM_STATUS will clear all of the bits in the command 48 communications status retry and abort masks. This will cause the system to ignore all bits in the communications status value. Communications status is defined to be the first data byte returned in a HART transaction, when bit 7 of this byte is set.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will be valid only during the current method. See send_command for implementation of the masks. See ABORT_ON_RESPONSE_CODE for a list of the available masks, and their default values.

The mask affected by this function is used by the send, send_trans, ext_send, ext_send_trans, get_more_status, and display functions.

Lexical structure

XMTR_RETRY_ON_ALL_COMM_STATUS(VOID)

The lexical element of the Builtin XMTR_RETRY_ON_ALL_COMM_STATUS is shown in Table 355.

Table 355 – Builtin XMTR_RETRY_ON_ALL_COMM_STATUS

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.352 Builtin XMTR_RETRY_ON_ALL_DATA

Purpose

The Builtin XMTR_RETRY_ON_ALL_DATA will set all of the bits in the command 48 device status retry mask. This will cause the system to retry if any bit in command 48 is set.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will be valid only during the current method. See send_command for implementation of the masks. See ABORT_ON_RESPONSE_CODE for a list of the available masks, and their default values.

The mask affected by this function is used by the send, send_trans, ext_send, ext_send_trans, get_more_status, and display functions.

Lexical structure

XMTR_RETRY_ON_ALL_DATA(VOID)

The lexical element of the Builtin XMTR_RETRY_ON_ALL_DATA is shown in Table 356.

Table 356 – Builtin XMTR_RETRY_ON_ALL_DATA

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.353 Builtin XMTR_RETRY_ON_ALL_DEVICE_STATUS

Purpose

The Builtin XMTR_RETRY_ON_ALL_DEVICE_STATUS will set all of the bits in the command 48 device status retry mask. This will cause the system to retry command 48 if the device returns any command 48 device status value. Device status is defined to be the second data octet returned in a transaction.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_RETRY_ON_ALL_DEVICE_STATUS(VOID)

The lexical element of the Builtin XMTR_RETRY_ON_ALL_DEVICE_STATUS is shown in Table 357.

Table 357 – Builtin XMTR_RETRY_ON_ALL_DEVICE_STATUS

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.354 Builtin XMTR_RETRY_ON_ALL_RESPONSE_CODE (removed)

4.355 Builtin XMTR_RETRY_ON_ALL_RESPONSE_CODES

Purpose

The Builtin XMTR_RETRY_ON_ALL_RESPONSE_CODE will set all of the bits in the command 48 response code retry mask. This will cause the system command 48 to be retried if the device returns any command 48 response code value.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_RETRY_ON_ALL_RESPONSE_CODES (VOID)

The lexical element of the Builtin XMTR_RETRY_ON_ALL_RESPONSE_CODES is shown in Table 358.

Table 358 – Builtin XMTR_RETRY_ON_ALL_RESPONSE_CODES

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.356 Builtin XMTR_RETRY_ON_COMM_ERROR

Purpose

The Builtin XMTR_RETRY_ON_COMM_ERROR will set the command 48 no comm error mask such that command 48 will be retried if a comm error is found while sending command 48.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_RETRY_ON_COMM_ERROR (VOID)

The lexical element of the Builtin XMTR_RETRY_ON_COMM_ERROR is shown in Table 359.

Table 359 – Builtin XMTR_RETRY_ON_COMM_ERROR

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.357 Builtin XMTR_RETRY_ON_COMM_STATUS

Purpose

The Builtin XMTR_RETRY_ON_COMM_STATUS will set the correct bit(s) in the command 48 comm status retry mask in such a way that the specified command 48 comm status value will cause command 48 to be retried. Comm status is defined to be the first data octet returned in a transaction, when bit 7 of this is 1.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_RETRY_ON_COMM_STATUS(comm_status)

The lexical elements of the Builtin XMTR_RETRY_ON_COMM_STATUS are shown in Table 360.

Table 360 – Builtin XMTR_RETRY_ON_COMM_STATUS

Parameter name	Type	Direction	Usage	Description
comm_status	int	l	m	The new comm status retry mask.
<return>	VOID	—	—	—

4.358 Builtin XMTR_RETRY_ON_DATA

Purpose

The Builtin XMTR_RETRY_ON_DATA will set the specified bit(s) in the retry data mask in such a way that the command 48 will be retried if the specified bit is set in the data field returned by command 48.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, and display.

Lexical structure

XMTR_RETRY_ON_DATA(data, bit)

The lexical elements of the Builtin XMTR_RETRY_ON_DATA are shown in Table 361.

Table 361 – Builtin XMTR_RETRY_ON_DATA

Parameter name	Type	Direction	Usage	Description
data	int	l	m	Zero based index into command 48 data field.
bit	int	l	m	Bit number (0..7). Bit 0 is the least significant bit.
<return>	VOID	—	—	—

4.359 Builtin XMTR_RETRY_ON_DEVICE_STATUS

Purpose

The Builtin XMTR_RETRY_ON_DEVICE_STATUS will set the correct bit(s) in the command 48 device status retry mask in such a way that the specified command 48 device status value will cause command 48 to be retried. Device status is defined to be the second data octet returned in a transaction.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the send_command function for implementation of the masks. See Builtin ABORT_ON_RESPONSE_CODE for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins `send_command`, `send_command_trans`, `ext_send_command`, `ext_send_command_trans`, `get_more_status`, and `display`.

Lexical structure

`XMTR_RETRY_ON_DEVICE_STATUS(device_status)`

The lexical elements of the Builtin `XMTR_RETRY_ON_DEVICE_STATUS` are shown in Table 362.

Table 362 – Builtin XMTR_RETRY_ON_DEVICE_STATUS

Parameter name	Type	Direction	Usage	Description
<code>device_status</code>	INT4	I	m	The new device status retry mask.
<code><return></code>	VOID	—	—	—

4.360 Builtin XMTR_RETRY_ON_NO_DEVICE

Purpose

The Builtin `XMTR_RETRY_ON_NO_DEVICE` will set the command 48 no-device mask in such a way that command 48 will be retried if no device is found while sending command 48.

The retry and abort masks are reset to their values at the start of each method, so the new mask value will only be valid during the current method. See the `send_command` function for implementation of the masks. See Builtin `ABORT_ON_RESPONSE_CODE` for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins `send_command`, `send_command_trans`, `ext_send_command`, `ext_send_command_trans`, `get_more_status`, and `display`.

Lexical structure

`XMTR_RETRY_ON_NO_DEVICE(VOID)`

The lexical element of the Builtin `XMTR_RETRY_ON_NO_DEVICE` is shown in Table 363.

Table 363 – Builtin XMTR_RETRY_ON_NO_DEVICE

Parameter name	Type	Direction	Usage	Description
<code><return></code>	VOID	—	—	—

4.361 Builtin XMTR_RETRY_ON_RESPONSE_CODE

Purpose

The Builtin `XMTR_RETRY_ON_RESPONSE_CODE` will set the correct bit(s) in the response code retry mask in such a way that the specified command 48 response code value will cause command 48 to be retried. The response code is defined to be the first data octet returned in a transaction, when bit 7 of this octet is 0.

The retry and abort masks are reset to their default values at the start of each method, so the new mask value will only be valid during the current method. See the `send_command` function for implementation of the masks. See Builtin `ABORT_ON_RESPONSE_CODE` for a list of the available masks and their default values.

The mask affected by this Builtin is used by the Builtins `send_command`, `send_command_trans`, `ext_send_command`, `ext_send_command_trans`, `get_more_status`, and `display`.

Lexical structure

`XMTR_RETRY_ON_RESPONSE_CODE(response_code)`

The lexical element of the Builtin `XMTR_RETRY_ON_RESPONSE_CODE` is shown in Table 364.

Table 364 – Builtin `XMTR_RETRY_ON_RESPONSE_CODE`

Parameter name	Type	Direction	Usage	Description
<return>	VOID	—	—	—

5 Builtins return codes

Some Builtins have specific return codes. These return codes are specified in Table 366 and Table 367. The description of Table 366 and Table 367 is in Table 365.

Table 365 – Contents of the return codes description table

Column	Meaning
Value	Specifies a long integer returned from the Builtin.
Identifier	Specifies a reference to a returned long integer value.
Description	Specifies a description for the return code.

Table 366 – Return code descriptions

Value	Identifier	Description
0	BLTIN_SUCCESS	Success.
1801	BLTIN_NO_MEMORY	Out of memory.
1802	BLTIN_VAR_NOT_FOUND	Cannot find variable.
1803	BLTIN_BAD_ID	Specified item or member ID does not exist.
1804	BLTIN_NO_DATA_TO_SEND	No value to send to device.
1805	BLTIN_WRONG_DATA_TYPE	Wrong variable type for Builtin.
1806	BLTIN_NO_RESP_CODES	Item does not have response codes.
1807	BLTIN_BAD_METHOD_ID	Invalid method ID.
1808	BLTIN_BUFFER_TOO_SMALL	Supplied buffer for starting string is too small.
1809	BLTIN_CANNOT_READ_VARIABLE	Cannot read the specified variable.
1810	BLTIN_INVALID_PROMPT	Bad prompt string.
1811	BLTIN_NO_LANGUAGE_STRING	No valid string for current language.
1812	BLTIN_DDS_ERROR	A DDS error occurred.
1813	BLTIN_FAIL_RESPONSE_CODE	Response code received from the device caused Builtin to fail.
1814	BLTIN_FAIL_COMM_ERROR	Communication error caused Builtin to fail.
1815	BLTIN_NOT_IMPLEMENTED	Builtin function not yet implemented.
1816	BLTIN_BAD_ITEM_TYPE	Wrong item type for Builtin.
1817	BLTIN_VALUE_NOT_SET	A CLASS LOCAL variable has not been initialized before being used.
1818	BLTIN_BLOCK_NOT_FOUND	Specified block does not exist within the device.

Table 367 – Return code descriptions

Value	Identifier	Description
0	BI_SUCCESS	The Builtin completed successfully.
-1	BI_ERROR	An error occurred during the execution of the Builtin.
-2	BI_ABORT	The user aborted the task during the execution of the Builtin.
-3	BI_NO_DEVICE	There was no response from the device during the execution of the Builtin.
-4	BI_COMM_ERR	A communications error occurred during the execution of the Builtin.

Annex A (normative)

Profiles of Builtins

A.1 Conventions for profiles of Builtins

The conventions are specified in IEC 61804-3:–, Annex D.1.

A.2 Profiles for PROFIBUS and PROFINET

Table A.1 is the selection of Builtins specified in Clause 4 and used by the consortium PROFIBUS&PROFINET International (PI).

The Builtins are listed in alphabetical order.

Table A.1 – Builtin profile for PROFIBUS and PROFINET

Reference	Builtin Name	Presence	Constraints
4.4	_ERROR	YES	—
4.5	_TRACE	YES	—
4.6	_WARNING	YES	—
4.7	abort	YES	—
4.8	abort_on_all_comm_errors	NO	—
4.9	ABORT_ON_ALL_COMM_STATUS	NO	—
4.10	ABORT_ON_ALL_DEVICE_STATUS	NO	—
4.11	ABORT_ON_ALL_RESPONSE_CODES	NO	—
4.12	abort_on_all_response_codes	NO	—
4.13	ABORT_ON_COMM_ERROR	NO	—
4.14	abort_on_comm_error	NO	—
4.15	ABORT_ON_COMM_STATUS	NO	—
4.16	ABORT_ON_DEVICE_STATUS	NO	—
4.17	ABORT_ON_NO_DEVICE	NO	—
4.18	ABORT_ON_RESPONSE_CODE	NO	—
4.19	abort_on_response_code	NO	—
4.20	abortTransferPort	NO	—
4.21	abs	YES	—
4.22	ACKNOWLEDGE	YES	—
4.23	acknowledge	YES	—
4.24	acos	YES	—
4.25	add_abort_method (version A)	YES	—
4.26	add_abort_method (version B)	NO	—
4.27	AddTime	YES	—
4.28	asin	YES	—
4.29	assign	NO	—
4.30	assign_double	YES	—

Reference	Builtin Name	Presence	Constraints
4.31	assign_float	YES	—
4.32	assign_int	YES	—
4.33	assign_var	YES	—
4.34	assign2	NO	—
4.35	atan	YES	—
4.36	atof	YES	—
4.37	atoi	YES	—
4.38	browseIdentity	NO	—
4.39	BUILD_MESSAGE	YES	—
4.40	ByteToDouble	YES	—
4.41	ByteToFloat	YES	—
4.42	ByteToLong	YES	—
4.43	ByteToShort	YES	—
4.44	cbirt	YES	—
4.45	ceil	YES	—
4.46	closeTransferPort	NO	—
4.47	cos	YES	—
4.48	cosh	YES	—
4.49	dassign	YES	—
4.50	DATE_AND_TIME_VALUE_to_string	YES	—
4.51	Date_to_DayOfMonth	YES	—
4.52	DATE_to_days	YES	—
4.53	Date_to_Month	YES	—
4.54	DATE_to_string	YES	—
4.55	Date_To_Time	YES	—
4.56	Date_to_Year	YES	—
4.57	days_to_DATE	YES	—
4.58	DELAY	YES	—
4.59	delay	YES	—
4.60	DELAY_TIME	YES	—
4.61	delayfor	NO	—
4.62	delayfor2	NO	—
4.63	DICT_ID	NO	—
4.64	dictionary_string	YES	—
4.65	DiffTime	YES	—
4.66	discard_on_exit	YES	—
4.67	DISPLAY	YES	—
4.68	display	YES	—
4.69	display_bitenum	YES	—
4.70	display_builtin_error	NO	—
4.71	display_comm_error	NO	—
4.72	display_comm_status	NO	—
4.73	display_device_status	NO	—
4.74	display_dynamics	NO	—

Reference	Builtin Name	Presence	Constraints
4.75	display_dynamics2	NO	—
4.76	display_message	NO	—
4.77	display_message2	NO	—
4.78	display_response_code	NO	—
4.79	display_response_status	NO	—
4.80	display_xmtr_status	NO	—
4.81	DoubleToByte	YES	—
4.82	drand	YES	—
4.83	dseed	YES	—
4.84	edit_device_value	NO	—
4.85	edit_device_value2	NO	—
4.86	edit_local_value	NO	—
4.87	edit_local_value2	NO	—
4.88	exp	YES	—
4.89	ext_send_command	NO	—
4.90	ext_send_command_trans	NO	—
4.91	fail_on_all_comm_errors	NO	—
4.92	fail_on_all_response_codes	NO	—
4.93	fail_on_comm_error	NO	—
4.94	fail_on_response_code	NO	—
4.95	fassign	YES	—
4.96	fGetByte	NO	—
4.97	fgetval	YES	—
4.98	float_value	YES	—
4.99	FloatToByte	YES	—
4.100	floor	YES	—
4.101	fmod	YES	—
4.102	fpclassify	YES	—
4.103	From_DATE_AND_TIME_VALUE	YES	—
4.104	From_TIME_VALUE	YES	—
4.105	fsetval	YES	—
4.106	ftoa	YES	—
4.107	fvar_value	YES	—
4.108	get_acknowledgement	NO	—
4.109	get_acknowledgement2	NO	—
4.110	get_block_instance_by_object_index	NO	—
4.111	get_block_instance_by_tag	NO	—
4.112	get_block_instance_count	NO	—
4.113	get_comm_error	NO	—
4.114	get_comm_error_string	NO	—
4.115	get_date	NO	—
4.116	get_date_lelem	NO	—
4.117	get_date_lelem2	NO	—
4.118	get_date_value	NO	—

Reference	Builtin Name	Presence	Constraints
4.119	get_date_value2	NO	—
4.120	GET_DD_REVISION	YES	—
4.121	get_dds_error	NO	—
4.122	GET_DEV_VAR_VALUE	YES	—
4.123	get_dev_var_value	YES	—
4.124	GET_DEVICE_REVISION	YES	—
4.125	GET_DEVICE_TYPE	YES	—
4.126	get_dictionary_string	YES	—
4.127	get_double	NO	—
4.128	get_double_lelem	NO	—
4.129	get_double_lelem2	NO	—
4.130	get_double_value	NO	—
4.131	get_double_value2	NO	—
4.132	get_enum_string	YES	—
4.133	get_float	NO	—
4.134	get_float_lelem	NO	—
4.135	get_float_lelem2	NO	—
4.136	get_float_value	NO	—
4.137	get_float_value2	NO	—
4.138	GET_LOCAL_VAR_VALUE	YES	—
4.139	get_local_var_value	YES	—
4.140	GET_MANUFACTURER	YES	—
4.141	get_more_status	NO	—
4.142	get_resolve_status	NO	—
4.143	get_response_code	NO	—
4.144	get_response_code_string	NO	—
4.145	get_rspcode_string	NO	—
4.146	get_rspcode_string_by_id	YES	—
4.147	get_signed	NO	—
4.148	get_signed_lelem	NO	—
4.149	get_signed_lelem2	NO	—
4.150	get_signed_value	NO	—
4.151	get_signed_value2	NO	—
4.152	get_status_code_string	NO	—
4.153	get_status_string	NO	—
4.154	get_stddict_string	NO	—
4.155	get_string	NO	—
4.156	get_string_lelem	NO	—
4.157	get_string_lelem2	NO	—
4.158	get_string_value	NO	—
4.159	get_string_value2	NO	—
4.160	GET_TICK_COUNT	YES	—
4.161	get_transfer_status	NO	—
4.162	get_unsigned	NO	—

Reference	Builtin Name	Presence	Constraints
4.163	get_unsigned_lelem	NO	—
4.164	get_unsigned_lelem2	NO	—
4.165	get_unsigned_value	NO	—
4.166	get_unsigned_value2	NO	—
4.167	get_variable_string	YES	—
4.168	GetCurrentDate	YES	—
4.169	GetCurrentDateAndTime	YES	—
4.170	GetCurrentTime	YES	—
4.171	iassign	YES	—
4.172	igetval	YES	—
4.173	IGNORE_ALL_COMM_STATUS	NO	—
4.174	IGNORE_ALL_DEVICE_STATUS	NO	—
4.175	IGNORE_ALL_RESPONSE_CODES	NO	—
4.176	IGNORE_COMM_ERROR	NO	—
4.177	IGNORE_COMM_STATUS	NO	—
4.178	IGNORE_DEVICE_STATUS	NO	—
4.179	IGNORE_NO_DEVICE	YES	—
4.180	IGNORE_RESPONSE_CODE	NO	—
4.181	int_value	YES	—
4.182	is_NaN	YES	—
4.183	isetval	YES	—
4.184	isOffline	YES	—
4.185	ITEM_ID	NO	—
4.186	itoa (version A)	YES	—
4.187	itoa (version B)	NO	—
4.188	ivar_value	YES	—
4.189	lassign	YES	—
4.190	lgetval	NO	—
4.191	ListDeleteElementAt	YES	—
4.192	ListDeleteElementAt2	NO	—
4.193	ListInsert	YES	—
4.194	ListInsert2	NO	—
4.195	log	YES	—
4.196	LOG_MESSAGE	YES	—
4.197	log10	YES	—
4.198	log2	YES	—
4.199	long_value	YES	—
4.200	LongToByte	YES	—
4.201	lsetval	YES	—
4.202	lvar_value	YES	—
4.203	Make_Time	YES	—
4.204	MEMBER_ID	NO	—
4.205	MenuDisplay	YES	—
4.206	method_abort	NO	—

Reference	Builtin Name	Presence	Constraints
4.207	nan	YES	—
4.208	NaN_value	YES	—
4.209	nanf	YES	—
4.210	ObjectReference	YES	—
4.211	openTransferPort	NO	—
4.212	pop_abort_method	YES	—
4.213	pow	YES	—
4.214	process_abort	YES	—
4.215	push_abort_method	YES	—
4.216	put_date	NO	—
4.217	put_date_value	NO	—
4.218	put_date_value2	NO	—
4.219	put_double	NO	—
4.220	put_double_value	NO	—
4.221	put_double_value2	NO	—
4.222	put_float	NO	—
4.223	put_float_value	NO	—
4.224	put_float_value2	NO	—
4.225	PUT_MESSAGE	YES	—
4.226	put_message	YES	—
4.227	put_signed	NO	—
4.228	put_signed_value	NO	—
4.229	put_signed_value2	NO	—
4.230	put_string	NO	—
4.231	put_string_value	NO	—
4.232	put_string_value2	NO	—
4.233	put_unsigned	NO	—
4.234	put_unsigned_value	NO	—
4.235	put_unsigned_value2	NO	—
4.236	re_read_file	NO	—
4.237	re_write_file	NO	—
4.238	read_value	NO	—
4.239	read_value2	NO	—
4.240	ReadCommand	YES	—
4.241	readItemFromDevice	NO	—
4.242	remove_abort_method (version A)	YES	—
4.243	remove_abort_method (version B)	NO	—
4.244	remove_all_abort_methods	YES	—
4.245	resolve_array_ref	NO	—
4.246	resolve_array_ref2	NO	—
4.247	resolve_block_ref	NO	—
4.248	resolve_block_ref2	NO	—
4.248	resolve_list_ref	NO	—
4.250	resolve_local_ref	NO	—

Reference	Builtin Name	Presence	Constraints
4.251	resolve_local_ref2	NO	—
4.252	resolve_param_list_ref	NO	—
4.253	resolve_param_ref	NO	—
4.254	resolve_param_ref2	NO	—
4.255	resolve_record_ref	NO	—
4.256	resolve_record_ref2	NO	—
4.257	ret_double_value	NO	—
4.258	ret_double_value2	NO	—
4.259	ret_float_value	NO	—
4.260	ret_float_value2	NO	—
4.261	ret_signed_value	NO	—
4.262	ret_signed_value2	NO	—
4.263	ret_unsigned_value	NO	—
4.264	ret_unsigned_value2	NO	—
4.265	retry_on_all_comm_errors	NO	—
4.266	RETRY_ON_ALL_COMM_STATUS	NO	—
4.267	RETRY_ON_ALL_DEVICE_STATUS	NO	—
4.268	RETRY_ON_ALL_RESPONSE_CODES	NO	—
4.269	retry_on_all_response_codes	NO	—
4.270	RETRY_ON_COMM_ERROR	NO	—
4.271	retry_on_comm_error	NO	—
4.272	RETRY_ON_COMM_STATUS	NO	—
4.273	RETRY_ON_DEVICE_STATUS	NO	—
4.274	RETRY_ON_NO_DEVICE	YES	—
4.275	RETRY_ON_RESPONSE_CODE	NO	—
4.276	retry_on_response_code	NO	—
4.277	round	YES	—
4.278	save_on_exit	YES	—
4.279	save_values	YES	—
4.280	seconds_to_TIME_VALUE	YES	—
4.281	seconds_to_TIME_VALUE8	YES	—
4.282	SELECT_FROM_LIST	YES	—
4.283	select_from_list	YES	—
4.284	select_from_menu	NO	—
4.285	select_from_menu2	NO	—
4.286	send	NO	—
4.287	send_all_values	NO	—
4.288	send_command	NO	—
4.289	send_command_trans	NO	—
4.290	send_on_exit	YES	—
4.291	send_trans	NO	—
4.292	send_value	NO	—
4.293	send_value2	NO	—
4.294	SET_NUMBER_OF_RETRIES	YES	—

Reference	Builtin Name	Presence	Constraints
4.295	sgetval	YES	—
4.296	ShortToByte	YES	—
4.297	sin	YES	—
4.298	sinh	YES	—
4.299	sqrt	YES	—
4.300	ssetval	YES	—
4.301	strcmp	YES	—
4.302	strleft	YES	—
4.303	strlen	YES	—
4.304	strlwr	YES	—
4.305	strmid	YES	—
4.306	strright	YES	—
4.307	strstr	YES	—
4.308	strtrim	YES	—
4.309	strupr	YES	—
4.310	tan	YES	—
4.311	tanh	YES	—
4.312	Time_To_Date	YES	—
4.313	TIME_VALUE_to_Hour	YES	—
4.314	TIME_VALUE_to_Minute	YES	—
4.315	TIME_VALUE_to_Second	YES	—
4.316	TIME_VALUE_to_seconds	YES	—
4.317	TIME_VALUE_to_string	YES	—
4.318	timet_to_string	YES	—
4.319	timet_to_TIME_VALUE	YES	—
4.320	timet_to_TIME_VALUE8	YES	—
4.321	To_Date	YES	—
4.322	To_Date_and_Time	YES	—
4.323	To_Time	YES	—
4.324	To_TIME_VALUE	YES	—
4.325	To_TIME_VALUE8	YES	—
4.326	trunc	YES	—
4.327	VARID	YES	—
4.328	vassign	YES	—
4.329	WriteCommand	YES	—
4.330	writeltemToDevice	NO	—
4.331	XMTR_ABORT_ON_ALL_COMM_STATUS	NO	—
4.332	XMTR_ABORT_ON_ALL_DATA	NO	—
4.333	XMTR_ABORT_ON_ALL_DEVICE_STATUS	NO	—
4.334	XMTR_ABORT_ON_ALL_RESPONSE_CODES	NO	—
4.335	XMTR_ABORT_ON_COMM_ERROR	NO	—
4.336	XMTR_ABORT_ON_COMM_STATUS	NO	—
4.337	XMTR_ABORT_ON_DATA	NO	—
4.338	XMTR_ABORT_ON_DEVICE_STATUS	NO	—

Reference	Builtin Name	Presence	Constraints
4.339	XMTR_ABORT_ON_NO_DEVICE	NO	—
4.340	XMTR_ABORT_ON_RESPONSE_CODE	NO	—
4.341	XMTR_IGNORE_ALL_COMM_STATUS	NO	—
4.342	XMTR_IGNORE_ALL_DATA	NO	—
4.343	XMTR_IGNORE_ALL_DEVICE_STATUS	NO	—
4.344	XMTR_IGNORE_ALL_RESPONSE_CODES	NO	—
4.345	XMTR_IGNORE_COMM_ERROR	NO	—
4.346	XMTR_IGNORE_COMM_STATUS	NO	—
4.347	XMTR_IGNORE_DATA	NO	—
4.348	XMTR_IGNORE_DEVICE_STATUS	NO	—
4.349	XMTR_IGNORE_NO_DEVICE	NO	—
4.350	XMTR_IGNORE_RESPONSE_CODE	NO	—
4.351	XMTR_RETRY_ON_ALL_COMM_STATUS	NO	—
4.352	XMTR_RETRY_ON_ALL_DATA	NO	—
4.353	XMTR_RETRY_ON_ALL_DEVICE_STATUS	NO	—
4.355	XMTR_RETRY_ON_ALL_RESPONSE_CODES	NO	—
4.356	XMTR_RETRY_ON_COMM_ERROR	NO	—
4.357	XMTR_RETRY_ON_COMM_STATUS	NO	—
4.358	XMTR_RETRY_ON_DATA	NO	—
4.359	XMTR_RETRY_ON_DEVICE_STATUS	NO	—
4.360	XMTR_RETRY_ON_NO_DEVICE	NO	—
4.361	XMTR_RETRY_ON_RESPONSE_CODE	NO	—

A.3 Profiles for FOUNDATION™³ Fieldbus

Table A.2 is the selection of Builtins specified in Clause 4 and used by the consortium Fieldbus Foundation.

The Builtins are listed in alphabetical order.

Table A.2 – Builtin profile for FOUNDATION fieldbus

Reference	Builtin Name	Presence	Constraints
4.4	_ERROR	YES	—
4.5	_TRACE	YES	—
4.6	_WARNING	YES	—
4.7	abort	NO	—
4.8	abort_on_all_comm_errors	YES	—
4.9	ABORT_ON_ALL_COMM_STATUS	NO	—
4.10	ABORT_ON_ALL_DEVICE_STATUS	NO	—
4.11	ABORT_ON_ALL_RESPONSE_CODES	NO	—
4.12	abort_on_all_response_codes	YES	—
4.13	ABORT_ON_COMM_ERROR	NO	—
4.14	abort_on_comm_error	YES	—
4.15	ABORT_ON_COMM_STATUS	NO	—
4.16	ABORT_ON_DEVICE_STATUS	NO	—
4.17	ABORT_ON_NO_DEVICE	NO	—
4.18	ABORT_ON_RESPONSE_CODE	NO	—
4.19	abort_on_response_code	YES	—
4.20	abortTransferPort	NO	—
4.21	abs	YES	—
4.22	ACKNOWLEDGE	NO	—
4.23	acknowledge	NO	—
4.24	acos	YES	—
4.25	add_abort_method (version A)	NO	—
4.26	add_abort_method (version B)	YES	—
4.27	AddTime	YES	—
4.28	asin	YES	—
4.29	assign	YES	—
4.30	assign_double	NO	—
4.31	assign_float	NO	—
4.32	assign_int	NO	—
4.33	assign_var	NO	—
4.34	assign2	YES	—
4.35	atan	YES	—

³ FOUNDATION™ Fieldbus is the trade name of the consortium Fieldbus Foundation (non-profit organization). This information is given for the convenience of users of this standard and does not constitute an endorsement by IEC of the trade name holder or any of its products. Compliance to this profile does not require use of the trade name. Use of the trade names requires permission from the trade name holder.

Reference	Builtin Name	Presence	Constraints
4.36	atof	NO	—
4.37	atoi	NO	—
4.38	browseIdentity	NO	—
4.39	BUILD_MESSAGE	YES	—
4.40	ByteToDouble	NO	—
4.41	ByteToFloat	NO	—
4.42	ByteToLong	NO	—
4.43	ByteToShort	NO	—
4.44	cbrt	YES	—
4.45	ceil	YES	—
4.46	closeTransferPort	NO	—
4.47	cos	YES	—
4.48	cosh	YES	—
4.49	dassign	NO	—
4.50	DATE_AND_TIME_VALUE_to_string	NO	—
4.51	Date_to_DayOfMonth	NO	—
4.52	DATE_to_days	NO	—
4.53	Date_to_Month	NO	—
4.54	DATE_to_string	NO	—
4.55	Date_To_Time	NO	—
4.56	Date_to_Year	NO	—
4.57	days_to_DATE	NO	—
4.58	DELAY	NO	—
4.59	delay	NO	—
4.60	DELAY_TIME	NO	—
4.61	delayfor	YES	—
4.62	delayfor2	YES	—
4.63	DICT_ID	YES	—
4.64	dictionary_string	NO	—
4.65	DiffTime	YES	—
4.66	discard_on_exit	YES	—
4.67	DISPLAY	NO	—
4.68	display	NO	—
4.69	display_bitenum	NO	—
4.70	display_builtin_error	YES	—
4.71	display_comm_error	YES	—
4.72	display_comm_status	NO	—
4.73	display_device_status	NO	—
4.74	display_dynamics	YES	—
4.75	display_dynamics2	YES	—
4.76	display_message	YES	—
4.77	display_message2	YES	—
4.78	display_response_code	YES	—
4.79	display_response_status	NO	—

Reference	Builtin Name	Presence	Constraints
4.80	display_xmtr_status	NO	—
4.81	DoubleToByte	NO	—
4.82	drand	YES	—
4.83	dseed	YES	—
4.84	edit_device_value	YES	—
4.85	edit_device_value2	YES	—
4.86	edit_local_value	YES	—
4.87	edit_local_value2	YES	—
4.88	exp	YES	—
4.89	ext_send_command	NO	—
4.90	ext_send_command_trans	NO	—
4.91	fail_on_all_comm_errors	YES	—
4.92	fail_on_all_response_codes	YES	—
4.93	fail_on_comm_error	YES	—
4.94	fail_on_response_code	YES	—
4.95	fassign	NO	—
4.96	fGetByte	NO	—
4.97	fgetval	NO	—
4.98	float_value	NO	—
4.99	FloatToByte	NO	—
4.100	floor	YES	—
4.101	fmod	YES	—
4.102	fpclassify	YES	—
4.103	From_DATE_AND_TIME_VALUE	NO	—
4.104	From_TIME_VALUE	NO	—
4.105	fsetval	NO	—
4.106	ftoa	NO	—
4.107	fvar_value	NO	—
4.108	get_acknowledgement	YES	—
4.109	get_acknowledgement2	YES	—
4.110	get_block_instance_by_object_index	YES	—
4.111	get_block_instance_by_tag	YES	—
4.112	get_block_instance_count	YES	—
4.113	get_comm_error	YES	—
4.114	get_comm_error_string	YES	—
4.115	get_date	YES	—
4.116	get_date_lelem	YES	—
4.117	get_date_lelem2	YES	—
4.118	get_date_value	YES	—
4.119	get_date_value2	YES	—
4.120	GET_DD_REVISION	NO	—
4.121	get_dds_error	YES	—
4.122	GET_DEV_VAR_VALUE	NO	—
4.123	get_dev_var_value	NO	—

Reference	Builtin Name	Presence	Constraints
4.124	GET_DEVICE_REVISION	NO	—
4.125	GET_DEVICE_TYPE	NO	—
4.126	get_dictionary_string	NO	—
4.127	get_double	YES	—
4.128	get_double_lelem	YES	—
4.129	get_double_lelem2	YES	—
4.130	get_double_value	YES	—
4.131	get_double_value2	YES	—
4.132	get_enum_string	NO	—
4.133	get_float	YES	—
4.134	get_float_lelem	YES	—
4.135	get_float_lelem2	YES	—
4.136	get_float_value	YES	—
4.137	get_float_value2	YES	—
4.138	GET_LOCAL_VAR_VALUE	NO	—
4.139	get_local_var_value	NO	—
4.140	GET_MANUFACTURER	NO	—
4.141	get_more_status	NO	—
4.142	get_resolve_status	YES	—
4.143	get_response_code	YES	—
4.144	get_response_code_string	YES	—
4.145	get_rspcode_string	NO	—
4.146	get_rspcode_string_by_id	NO	—
4.147	get_signed	YES	—
4.148	get_signed_lelem	YES	—
4.149	get_signed_lelem2	YES	—
4.150	get_signed_value	YES	—
4.151	get_signed_value2	YES	—
4.152	get_status_code_string	NO	—
4.153	get_status_string	YES	—
4.154	get_stddict_string	YES	—
4.155	get_string	YES	—
4.156	get_string_lelem	YES	—
4.157	get_string_lelem2	YES	—
4.158	get_string_value	YES	—
4.159	get_string_value2	YES	—
4.160	GET_TICK_COUNT	NO	—
4.161	get_transfer_status	NO	—
4.162	get_unsigned	YES	—
4.163	get_unsigned_lelem	YES	—
4.164	get_unsigned_lelem2	YES	—
4.165	get_unsigned_value	YES	—
4.166	get_unsigned_value2	YES	—
4.167	get_variable_string	NO	—

Reference	Builtin Name	Presence	Constraints
4.168	GetCurrentDate	YES	—
4.169	GetCurrentDateAndTime	YES	—
4.170	GetCurrentTime	YES	—
4.171	iassign	NO	—
4.172	igetval	NO	—
4.173	IGNORE_ALL_COMM_STATUS	NO	—
4.174	IGNORE_ALL_DEVICE_STATUS	NO	—
4.175	IGNORE_ALL_RESPONSE_CODES	NO	—
4.176	IGNORE_COMM_ERROR	NO	—
4.177	IGNORE_COMM_STATUS	NO	—
4.178	IGNORE_DEVICE_STATUS	NO	—
4.179	IGNORE_NO_DEVICE	NO	—
4.180	IGNORE_RESPONSE_CODE	NO	—
4.181	int_value	NO	—
4.182	is_NaN	YES	—
4.183	isetval	NO	—
4.184	isOffline	YES	—
4.185	ITEM_ID	YES	—
4.186	itoa (version A)	NO	—
4.187	itoa (version B)	NO	—
4.188	ivar_value	NO	—
4.189	lassign	NO	—
4.190	lgetval	NO	—
4.191	ListDeleteElementAt	YES	—
4.192	ListDeleteElementAt2	YES	—
4.193	ListInsert	YES	—
4.194	ListInsert2	YES	—
4.195	log	YES	—
4.196	LOG_MESSAGE	NO	—
4.197	log10	YES	—
4.198	log2	YES	—
4.199	long_value	NO	—
4.200	LongToByte	NO	—
4.201	lsetval	NO	—
4.202	lvar_value	NO	—
4.203	Make_Time	YES	—
4.204	MEMBER_ID	YES	—
4.205	MenuDisplay	YES	—
4.206	method_abort	YES	—
4.207	nan	YES	—
4.208	NaN_value	YES	—
4.209	nanf	YES	—
4.210	ObjectReference	NO	—
4.211	openTransferPort	NO	—

Reference	Builtin Name	Presence	Constraints
4.212	pop_abort_method	NO	—
4.213	pow	YES	—
4.214	process_abort	NO	—
4.215	push_abort_method	NO	—
4.216	put_date	YES	—
4.217	put_date_value	YES	—
4.218	put_date_value2	YES	—
4.219	put_double	YES	—
4.220	put_double_value	YES	—
4.221	put_double_value2	YES	—
4.222	put_float	YES	—
4.223	put_float_value	YES	—
4.224	put_float_value2	YES	—
4.225	PUT_MESSAGE	NO	—
4.226	put_message	NO	—
4.227	put_signed	YES	—
4.228	put_signed_value	YES	—
4.229	put_signed_value2	YES	—
4.230	put_string	YES	—
4.231	put_string_value	YES	—
4.232	put_string_value2	YES	—
4.233	put_unsigned	YES	—
4.234	put_unsigned_value	YES	—
4.235	put_unsigned_value2	YES	—
4.236	re_read_file	NO	—
4.237	re_write_file	NO	—
4.238	read_value	YES	—
4.239	read_value2	YES	—
4.240	ReadCommand	NO	—
4.241	readItemFromDevice	NO	—
4.242	remove_abort_method (version A)	NO	—
4.243	remove_abort_method (version B)	YES	—
4.244	remove_all_abort_methods	YES	—
4.245	resolve_array_ref	YES	—
4.246	resolve_array_ref2	YES	—
4.247	resolve_block_ref	YES	—
4.248	resolve_block_ref2	YES	—
4.249	resolve_list_ref	YES	—
4.250	resolve_local_ref	YES	—
4.251	resolve_local_ref2	YES	—
4.252	resolve_param_list_ref	YES	—
4.253	resolve_param_ref	YES	—
4.254	resolve_param_ref2	YES	—
4.255	resolve_record_ref	YES	—

Reference	Builtin Name	Presence	Constraints
4.256	resolve_record_ref2	YES	—
4.257	ret_double_value	YES	—
4.258	ret_double_value2	YES	—
4.259	ret_float_value	YES	—
4.260	ret_float_value2	YES	—
4.261	ret_signed_value	YES	—
4.262	ret_signed_value2	YES	—
4.263	ret_unsigned_value	YES	—
4.264	ret_unsigned_value2	YES	—
4.265	retry_on_all_comm_errors	YES	—
4.266	RETRY_ON_ALL_COMM_STATUS	NO	—
4.267	RETRY_ON_ALL_DEVICE_STATUS	NO	—
4.268	RETRY_ON_ALL_RESPONSE_CODES	NO	—
4.269	retry_on_all_response_codes	YES	—
4.270	RETRY_ON_COMM_ERROR	NO	—
4.271	retry_on_comm_error	YES	—
4.272	RETRY_ON_COMM_STATUS	NO	—
4.273	RETRY_ON_DEVICE_STATUS	NO	—
4.274	RETRY_ON_NO_DEVICE	NO	—
4.275	RETRY_ON_RESPONSE_CODE	NO	—
4.276	retry_on_response_code	YES	—
4.277	round	YES	—
4.278	save_on_exit	YES	—
4.279	save_values	NO	—
4.280	seconds_to_TIME_VALUE	NO	—
4.281	seconds_to_TIME_VALUE8	YES	—
4.282	SELECT_FROM_LIST	NO	—
4.283	select_from_list	NO	—
4.284	select_from_menu	YES	—
4.285	select_from_menu2	YES	—
4.286	send	NO	—
4.287	send_all_values	YES	—
4.288	send_command	NO	—
4.289	send_command_trans	NO	—
4.290	send_on_exit	YES	—
4.291	send_trans	NO	—
4.292	send_value	YES	—
4.293	send_value2	YES	—
4.294	SET_NUMBER_OF_RETRIES	NO	—
4.295	sgetval	NO	—
4.296	ShortToByte	NO	—
4.297	sin	YES	—
4.298	sinh	YES	—
4.299	sqrt	YES	—

Reference	Builtin Name	Presence	Constraints
4.300	ssetval	NO	—
4.301	strcmp	YES	—
4.302	strleft	YES	—
4.303	strlen	YES	—
4.304	strlwr	YES	—
4.305	strmid	YES	—
4.306	strright	YES	—
4.307	strstr	YES	—
4.308	strtrim	YES	—
4.309	strupr	YES	—
4.310	tan	YES	—
4.311	tanh	YES	—
4.312	Time_To_Date	NO	—
4.313	TIME_VALUE_to_Hour	YES	—
4.314	TIME_VALUE_to_Minute	YES	—
4.315	TIME_VALUE_to_Second	YES	—
4.316	TIME_VALUE_to_seconds	YES	—
4.317	TIME_VALUE_to_string	YES	—
4.318	timet_to_string	YES	—
4.319	timet_to_TIME_VALUE	NO	—
4.320	timet_to_TIME_VALUE8	YES	—
4.321	To_Date	NO	—
4.322	To_Date_and_Time	NO	—
4.323	To_Time	NO	—
4.324	To_TIME_VALUE	NO	—
4.325	To_TIME_VALUE8	YES	—
4.326	trunc	YES	—
4.327	VARID	NO	—
4.328	vassign	NO	—
4.329	WriteCommand	NO	—
4.330	writeItemToDevice	NO	—
4.331	XMTR_ABORT_ON_ALL_COMM_STATUS	NO	—
4.332	XMTR_ABORT_ON_ALL_DATA	NO	—
4.333	XMTR_ABORT_ON_ALL_DEVICE_STATUS	NO	—
4.334	XMTR_ABORT_ON_ALL_RESPONSE_CODES	NO	—
4.335	XMTR_ABORT_ON_COMM_ERROR	NO	—
4.336	XMTR_ABORT_ON_COMM_STATUS	NO	—
4.337	XMTR_ABORT_ON_DATA	NO	—
4.338	XMTR_ABORT_ON_DEVICE_STATUS	NO	—
4.339	XMTR_ABORT_ON_NO_DEVICE	NO	—
4.340	XMTR_ABORT_ON_RESPONSE_CODE	NO	—
4.341	XMTR_IGNORE_ALL_COMM_STATUS	NO	—
4.342	XMTR_IGNORE_ALL_DATA	NO	—
4.343	XMTR_IGNORE_ALL_DEVICE_STATUS	NO	—

Reference	Builtin Name	Presence	Constraints
4.344	XMTR_IGNORE_ALL_RESPONSE_CODES	NO	—
4.345	XMTR_IGNORE_COMM_ERROR	NO	—
4.346	XMTR_IGNORE_COMM_STATUS	NO	—
4.347	XMTR_IGNORE_DATA	NO	—
4.348	XMTR_IGNORE_DEVICE_STATUS	NO	—
4.349	XMTR_IGNORE_NO_DEVICE	NO	—
4.350	XMTR_IGNORE_RESPONSE_CODE	NO	—
4.351	XMTR_RETRY_ON_ALL_COMM_STATUS	NO	—
4.352	XMTR_RETRY_ON_ALL_DATA	NO	—
4.353	XMTR_RETRY_ON_ALL_DEVICE_STATUS	NO	—
4.355	XMTR_RETRY_ON_ALL_RESPONSE_CODES	NO	—
4.356	XMTR_RETRY_ON_COMM_ERROR	NO	—
4.357	XMTR_RETRY_ON_COMM_STATUS	NO	—
4.358	XMTR_RETRY_ON_DATA	NO	—
4.359	XMTR_RETRY_ON_DEVICE_STATUS	NO	—
4.360	XMTR_RETRY_ON_NO_DEVICE	NO	—
4.361	XMTR_RETRY_ON_RESPONSE_CODE	NO	—

A.4 Profiles for HART®

Table A.3 is the selection of Builtins specified in Clause 4 and used by the consortium HART Communication Foundation (HCF).

The Builtins are listed in alphabetical order.

Table A.3 – Builtin profile for HART

Reference	Builtin Name	Presence	Constraints
4.4	_ERROR	YES	—
4.5	_TRACE	YES	—
4.6	_WARNING	YES	—
4.7	abort	YES	—
4.8	abort_on_all_comm_errors	NO	—
4.9	ABORT_ON_ALL_COMM_STATUS	YES	—
4.10	ABORT_ON_ALL_DEVICE_STATUS	YES	—
4.11	ABORT_ON_ALL_RESPONSE_CODES	YES	—
4.12	abort_on_all_response_codes	NO	—
4.13	ABORT_ON_COMM_ERROR	YES	—
4.14	abort_on_comm_error	NO	—
4.15	ABORT_ON_COMM_STATUS	YES	—
4.16	ABORT_ON_DEVICE_STATUS	YES	—
4.17	ABORT_ON_NO_DEVICE	YES	—
4.18	ABORT_ON_RESPONSE_CODE	YES	—
4.19	abort_on_response_code	NO	—
4.20	abortTransferPort	YES	—

Reference	Builtin Name	Presence	Constraints
4.21	abs	YES	—
4.22	ACKNOWLEDGE	YES	—
4.23	acknowledge	YES	—
4.24	acos	YES	—
4.25	add_abort_method (version A)	YES	—
4.26	add_abort_method (version B)	NO	—
4.27	AddTime	YES	—
4.28	asin	YES	—
4.29	assign	NO	—
4.30	assign_double	YES	—
4.31	assign_float	YES	—
4.32	assign_int	YES	—
4.33	assign_var	YES	—
4.34	assign2	NO	—
4.35	atan	YES	—
4.36	atof	YES	—
4.37	atoi	YES	—
4.38	browseIdentity	YES	—
4.39	BUILD_MESSAGE	YES	—
4.40	ByteToDouble	NO	—
4.41	ByteToFloat	NO	—
4.42	ByteToLong	NO	—
4.43	ByteToShort	NO	—
4.44	cbrt	YES	—
4.45	ceil	YES	—
4.46	closeTransferPort	YES	—
4.47	cos	YES	—
4.48	cosh	YES	—
4.49	dassign	YES	—
4.50	DATE_AND_TIME_VALUE_to_string	YES	—
4.51	Date_to_DayOfMonth	YES	—
4.52	DATE_to_days	YES	—
4.53	Date_to_Month	YES	—
4.54	DATE_to_string	YES	—
4.55	Date_To_Time	YES	—
4.56	Date_to_Year	YES	—
4.57	days_to_DATE	YES	—
4.58	DELAY	YES	—
4.59	delay	YES	—
4.60	DELAY_TIME	YES	—
4.61	delayfor	NO	—
4.62	delayfor2	NO	—
4.63	DICT_ID	NO	—
4.64	dictionary_string	YES	—

Reference	Builtin Name	Presence	Constraints
4.65	DiffTime	YES	—
4.66	discard_on_exit	YES	—
4.67	DISPLAY	YES	—
4.68	display	YES	—
4.69	display_bitenum	NO	—
4.70	display_builtin_error	NO	—
4.71	display_comm_error	NO	—
4.72	display_comm_status	YES	—
4.73	display_device_status	YES	—
4.74	display_dynamics	NO	—
4.75	display_dynamics2	NO	—
4.76	display_message	NO	—
4.77	display_message2	NO	—
4.78	display_response_code	NO	—
4.79	display_response_status	YES	—
4.80	display_xmtr_status	YES	—
4.81	DoubleToByte	NO	—
4.82	drand	YES	—
4.83	dseed	YES	—
4.84	edit_device_value	NO	—
4.85	edit_device_value2	NO	—
4.86	edit_local_value	NO	—
4.87	edit_local_value2	NO	—
4.88	exp	YES	—
4.89	ext_send_command	YES	—
4.90	ext_send_command_trans	YES	—
4.91	fail_on_all_comm_errors	NO	—
4.92	fail_on_all_response_codes	NO	—
4.93	fail_on_comm_error	NO	—
4.94	fail_on_response_code	NO	—
4.95	fassign	YES	—
4.96	fGetByte	YES	—
4.97	fgetval	YES	—
4.98	float_value	YES	—
4.99	FloatToByte	NO	—
4.100	floor	YES	—
4.101	fmod	YES	—
4.102	fpclassify	YES	—
4.103	From_DATE_AND_TIME_VALUE	YES	—
4.104	From_TIME_VALUE	YES	—
4.105	fsetval	YES	—
4.106	ftoa	NO	—
4.107	fvar_value	YES	—
4.108	get_acknowledgement	NO	—

Reference	Builtin Name	Presence	Constraints
4.109	get_acknowledgement2	NO	—
4.110	get_block_instance_by_object_index	NO	—
4.111	get_block_instance_by_tag	NO	—
4.112	get_block_instance_count	NO	—
4.113	get_comm_error	NO	—
4.114	get_comm_error_string	NO	—
4.115	get_date	NO	—
4.116	get_date_lelem	NO	—
4.117	get_date_lelem2	NO	—
4.118	get_date_value	NO	—
4.119	get_date_value2	NO	—
4.120	GET_DD_REVISION	NO	—
4.121	get_dds_error	NO	—
4.122	GET_DEV_VAR_VALUE	YES	—
4.123	get_dev_var_value	YES	—
4.124	GET_DEVICE_REVISION	NO	—
4.125	GET_DEVICE_TYPE	NO	—
4.126	get_dictionary_string	YES	—
4.127	get_double	NO	—
4.128	get_double_lelem	NO	—
4.129	get_double_lelem2	NO	—
4.130	get_double_value	NO	—
4.131	get_double_value2	NO	—
4.132	get_enum_string	YES	—
4.133	get_float	NO	—
4.134	get_float_lelem	NO	—
4.135	get_float_lelem2	NO	—
4.136	get_float_value	NO	—
4.137	get_float_value2	NO	—
4.138	GET_LOCAL_VAR_VALUE	YES	—
4.139	get_local_var_value	YES	—
4.140	GET_MANUFACTURER	NO	—
4.141	get_more_status	YES	—
4.142	get_resolve_status	NO	—
4.143	get_response_code	NO	—
4.144	get_response_code_string	NO	—
4.145	get_rspcode_string	YES	—
4.146	get_rspcode_string_by_id	NO	—
4.147	get_signed	NO	—
4.148	get_signed_lelem	NO	—
4.149	get_signed_lelem2	NO	—
4.150	get_signed_value	NO	—
4.151	get_signed_value2	NO	—
4.152	get_status_code_string	YES	—

Reference	Builtin Name	Presence	Constraints
4.153	get_status_string	NO	—
4.154	get_stddict_string	NO	—
4.155	get_string	NO	—
4.156	get_string_lelem	NO	—
4.157	get_string_lelem2	NO	—
4.158	get_string_value	NO	—
4.159	get_string_value2	NO	—
4.160	GET_TICK_COUNT	NO	—
4.161	get_transfer_status	YES	—
4.162	get_unsigned	NO	—
4.163	get_unsigned_lelem	NO	—
4.164	get_unsigned_lelem2	NO	—
4.165	get_unsigned_value	NO	—
4.166	get_unsigned_value2	NO	—
4.167	get_variable_string	NO	—
4.168	GetCurrentDate	NO	—
4.169	GetCurrentDateAndTime	NO	—
4.170	GetCurrentTime	YES	—
4.171	lassign	YES	—
4.172	lgetval	YES	—
4.173	IGNORE_ALL_COMM_STATUS	YES	—
4.174	IGNORE_ALL_DEVICE_STATUS	YES	—
4.175	IGNORE_ALL_RESPONSE_CODES	YES	—
4.176	IGNORE_COMM_ERROR	YES	—
4.177	IGNORE_COMM_STATUS	YES	—
4.178	IGNORE_DEVICE_STATUS	YES	—
4.179	IGNORE_NO_DEVICE	YES	—
4.180	IGNORE_RESPONSE_CODE	YES	—
4.181	int_value	YES	—
4.182	is_NaN	NO	—
4.183	lsetval	YES	—
4.184	isOffline	YES	—
4.185	ITEM_ID	NO	—
4.186	itoa (version A)	NO	—
4.187	itoa (version B)	YES	—
4.188	ivar_value	YES	—
4.189	Lassign	NO	—
4.190	Lgetval	NO	—
4.191	ListDeleteElementAt	YES	—
4.192	ListDeleteElementAt2	NO	—
4.193	ListInsert	YES	—
4.194	ListInsert2	NO	—
4.195	Log	YES	—
4.196	LOG_MESSAGE	YES	—

Reference	Builtin Name	Presence	Constraints
4.197	log10	YES	—
4.198	log2	YES	—
4.199	long_value	NO	—
4.200	LongToByte	NO	—
4.201	Lsetval	NO	—
4.202	lvar_value	NO	—
4.203	Make_Time	YES	—
4.204	MEMBER_ID	NO	—
4.205	MenuDisplay	YES	—
4.206	method_abort	NO	—
4.207	Nan	YES	—
4.208	NaN_value	NO	—
4.209	Nanf	YES	—
4.210	ObjectReference	NO	—
4.211	openTransferPort	YES	—
4.212	pop_abort_method	YES	—
4.213	pow	YES	—
4.214	process_abort	YES	—
4.215	push_abort_method	YES	—
4.216	put_date	NO	—
4.217	put_date_value	NO	—
4.218	put_date_value2	NO	—
4.219	put_double	NO	—
4.220	put_double_value	NO	—
4.221	put_double_value2	NO	—
4.222	put_float	NO	—
4.223	put_float_value	NO	—
4.224	put_float_value2	NO	—
4.225	PUT_MESSAGE	YES	—
4.226	put_message	YES	—
4.227	put_signed	NO	—
4.228	put_signed_value	NO	—
4.229	put_signed_value2	NO	—
4.230	put_string	NO	—
4.231	put_string_value	NO	—
4.232	put_string_value2	NO	—
4.233	put_unsigned	NO	—
4.234	put_unsigned_value	NO	—
4.235	put_unsigned_value2	NO	—
4.236	re_read_file	YES	—
4.237	re_write_file	YES	—
4.238	read_value	NO	—
4.239	read_value2	NO	—
4.240	ReadCommand	NO	—

Reference	Builtin Name	Presence	Constraints
4.241	readItemFromDevice	YES	—
4.242	remove_abort_method (version A)	YES	—
4.243	remove_abort_method (version B)	NO	—
4.244	remove_all_abort_methods	YES	—
4.245	resolve_array_ref	NO	—
4.246	resolve_array_ref2	NO	—
4.247	resolve_block_ref	NO	—
4.248	resolve_block_ref2	NO	—
4.249	resolve_list_ref	NO	—
4.250	resolve_local_ref	NO	—
4.251	resolve_local_ref2	NO	—
4.252	resolve_param_list_ref	NO	—
4.253	resolve_param_ref	NO	—
4.254	resolve_param_ref2	NO	—
4.255	resolve_record_ref	NO	—
4.256	resolve_record_ref2	NO	—
4.257	ret_double_value	NO	—
4.258	ret_double_value2	NO	—
4.259	ret_float_value	NO	—
4.260	ret_float_value2	NO	—
4.261	ret_signed_value	NO	—
4.262	ret_signed_value2	NO	—
4.263	ret_unsigned_value	NO	—
4.264	ret_unsigned_value2	NO	—
4.265	retry_on_all_comm_errors	NO	—
4.266	RETRY_ON_ALL_COMM_STATUS	YES	—
4.267	RETRY_ON_ALL_DEVICE_STATUS	YES	—
4.268	RETRY_ON_ALL_RESPONSE_CODES	YES	—
4.269	retry_on_all_response_codes	NO	—
4.270	RETRY_ON_COMM_ERROR	YES	—
4.271	retry_on_comm_error	NO	—
4.272	RETRY_ON_COMM_STATUS	YES	—
4.273	RETRY_ON_DEVICE_STATUS	YES	—
4.274	RETRY_ON_NO_DEVICE	YES	—
4.275	RETRY_ON_RESPONSE_CODE	YES	—
4.276	retry_on_response_code	NO	—
4.277	round	YES	—
4.278	save_on_exit	NO	—
4.279	save_values	YES	—
4.280	seconds_to_TIME_VALUE	YES	—
4.281	seconds_to_TIME_VALUE8	NO	—
4.282	SELECT_FROM_LIST	YES	—
4.283	select_from_list	YES	—
4.284	select_from_menu	NO	—

Reference	Builtin Name	Presence	Constraints
4.285	select_from_menu2	NO	—
4.286	send	YES	—
4.287	send_all_values	NO	—
4.288	send_command	YES	—
4.289	send_command_trans	YES	—
4.290	send_on_exit	NO	—
4.291	send_trans	YES	—
4.292	send_value	NO	—
4.293	send_value2	NO	—
4.294	SET_NUMBER_OF_RETRIES	YES	—
4.295	sgetval	YES	—
4.296	ShortToByte	NO	—
4.297	sin	YES	—
4.298	sinh	YES	—
4.299	sqrt	YES	—
4.300	ssetval	YES	—
4.301	strcmp	YES	—
4.302	strleft	NO	—
4.303	strlen	YES	—
4.304	strlwr	YES	—
4.305	strmid	YES	—
4.306	strright	NO	—
4.307	strstr	YES	—
4.308	strtrim	YES	—
4.309	strupr	YES	—
4.310	tan	YES	—
4.311	tanh	YES	—
4.312	Time_To_Date	YES	—
4.313	TIME_VALUE_to_Hour	YES	—
4.314	TIME_VALUE_to_Minute	YES	—
4.315	TIME_VALUE_to_Second	YES	—
4.316	TIME_VALUE_to_seconds	YES	—
4.317	TIME_VALUE_to_string	YES	—
4.318	timet_to_string	YES	—
4.319	timet_to_TIME_VALUE	YES	—
4.320	timet_to_TIME_VALUE8	NO	—
4.321	To_Date	YES	—
4.322	To_Date_and_Time	NO	—
4.323	To_Time	YES	—
4.324	To_TIME_VALUE	YES	—
4.325	To_TIME_VALUE8	NO	—
4.326	trunc	YES	—
4.327	VARID	YES	—
4.328	vassign	YES	—

Reference	Builtin Name	Presence	Constraints
4.329	WriteCommand	NO	—
4.330	writeItemToDevice	YES	—
4.331	XMTR_ABORT_ON_ALL_COMM_STATUS	YES	—
4.332	XMTR_ABORT_ON_ALL_DATA	YES	—
4.333	XMTR_ABORT_ON_ALL_DEVICE_STATUS	YES	—
4.334	XMTR_ABORT_ON_ALL_RESPONSE_CODES	YES	—
4.335	XMTR_ABORT_ON_COMM_ERROR	YES	—
4.336	XMTR_ABORT_ON_COMM_STATUS	YES	—
4.337	XMTR_ABORT_ON_DATA	YES	—
4.338	XMTR_ABORT_ON_DEVICE_STATUS	YES	—
4.339	XMTR_ABORT_ON_NO_DEVICE	YES	—
4.340	XMTR_ABORT_ON_RESPONSE_CODE	YES	—
4.341	XMTR_IGNORE_ALL_COMM_STATUS	YES	—
4.342	XMTR_IGNORE_ALL_DATA	YES	—
4.343	XMTR_IGNORE_ALL_DEVICE_STATUS	YES	—
4.344	XMTR_IGNORE_ALL_RESPONSE_CODES	YES	—
4.345	XMTR_IGNORE_COMM_ERROR	YES	—
4.346	XMTR_IGNORE_COMM_STATUS	YES	—
4.347	XMTR_IGNORE_DATA	YES	—
4.348	XMTR_IGNORE_DEVICE_STATUS	YES	—
4.349	XMTR_IGNORE_NO_DEVICE	YES	—
4.350	XMTR_IGNORE_RESPONSE_CODE	YES	—
4.351	XMTR_RETRY_ON_ALL_COMM_STATUS	YES	—
4.352	XMTR_RETRY_ON_ALL_DATA	YES	—
4.353	XMTR_RETRY_ON_ALL_DEVICE_STATUS	YES	—
4.355	XMTR_RETRY_ON_ALL_RESPONSE_CODES	YES	—
4.356	XMTR_RETRY_ON_COMM_ERROR	YES	—
4.357	XMTR_RETRY_ON_COMM_STATUS	YES	—
4.358	XMTR_RETRY_ON_DATA	YES	—
4.359	XMTR_RETRY_ON_DEVICE_STATUS	YES	—
4.360	XMTR_RETRY_ON_NO_DEVICE	YES	—
4.361	XMTR_RETRY_ON_RESPONSE_CODE	YES	—

A.5 Profiles for Communication Servers

Table A.4 is the selection of Builtins specified in Clause 4 and used for Communication Servers.

The Builtins are listed in alphabetical order.

Table A.4 – Builtin profile for Communication Servers

Reference	Builtin Name	Presence	Constraints
4.4	_ERROR	YES	—
4.5	_TRACE	YES	—
4.6	_WARNING	YES	—
4.7	abort	YES	—
4.8	abort_on_all_comm_errors	NO	—
4.9	ABORT_ON_ALL_COMM_STATUS	NO	—
4.10	ABORT_ON_ALL_DEVICE_STATUS	NO	—
4.11	ABORT_ON_ALL_RESPONSE_CODES	NO	—
4.12	abort_on_all_response_codes	NO	—
4.13	ABORT_ON_COMM_ERROR	NO	—
4.14	abort_on_comm_error	NO	—
4.15	ABORT_ON_COMM_STATUS	NO	—
4.16	ABORT_ON_DEVICE_STATUS	NO	—
4.17	ABORT_ON_NO_DEVICE	NO	—
4.18	ABORT_ON_RESPONSE_CODE	NO	—
4.19	abort_on_response_code	NO	—
4.20	abortTransferPort	NO	—
4.21	abs	YES	—
4.22	ACKNOWLEDGE	YES	—
4.23	acknowledge	YES	—
4.24	acos	YES	—
4.25	add_abort_method (version A)	YES	—
4.26	add_abort_method (version B)	NO	—
4.27	AddTime	YES	—
4.28	asin	YES	—
4.29	assign	NO	—
4.30	assign_double	YES	—
4.31	assign_float	YES	—
4.32	assign_int	YES	—
4.33	assign_var	YES	—
4.34	assign2	NO	—
4.35	atan	YES	—
4.36	atof	YES	—
4.37	atoi	YES	—
4.38	browseIdentity	NO	—
4.39	BUILD_MESSAGE	YES	—
4.40	ByteToDouble	YES	—

Reference	Builtin Name	Presence	Constraints
4.41	ByteToFloat	YES	—
4.42	ByteToLong	YES	—
4.43	ByteToShort	YES	—
4.44	cbrt	YES	—
4.45	ceil	YES	—
4.46	closeTransferPort	NO	—
4.47	cos	YES	—
4.48	cosh	YES	—
4.49	dassign	YES	—
4.50	DATE_AND_TIME_VALUE_to_string	YES	—
4.51	Date_to_DayOfMonth	YES	—
4.52	DATE_to_days	YES	—
4.53	Date_to_Month	YES	—
4.54	DATE_to_string	YES	—
4.55	Date_To_Time	YES	—
4.56	Date_to_Year	YES	—
4.57	days_to_DATE	YES	—
4.58	DELAY	YES	—
4.59	delay	YES	—
4.60	DELAY_TIME	YES	—
4.61	delayfor	NO	—
4.62	delayfor2	NO	—
4.63	DICT_ID	NO	—
4.64	dictionary_string	YES	—
4.65	DiffTime	YES	—
4.66	discard_on_exit	YES	—
4.67	DISPLAY	YES	—
4.68	display	YES	—
4.69	display_bitenum	YES	—
4.70	display_builtin_error	NO	—
4.71	display_comm_error	NO	—
4.72	display_comm_status	NO	—
4.73	display_device_status	NO	—
4.74	display_dynamics	NO	—
4.75	display_dynamics2	NO	—
4.76	display_message	NO	—
4.77	display_message2	NO	—
4.78	display_response_code	NO	—
4.79	display_response_status	NO	—
4.80	display_xmtr_status	NO	—
4.81	DoubleToByte	YES	—
4.82	drand	YES	—
4.83	dseed	YES	—
4.84	edit_device_value	NO	—

Reference	Builtin Name	Presence	Constraints
4.85	edit_device_value2	NO	—
4.86	edit_local_value	NO	—
4.87	edit_local_value2	NO	—
4.88	exp	YES	—
4.89	ext_send_command	NO	—
4.90	ext_send_command_trans	NO	—
4.91	fail_on_all_comm_errors	NO	—
4.92	fail_on_all_response_codes	NO	—
4.93	fail_on_comm_error	NO	—
4.94	fail_on_response_code	NO	—
4.95	fassign	YES	—
4.96	fGetByte	NO	—
4.97	fgetval	YES	—
4.98	float_value	YES	—
4.99	FloatToByte	YES	—
4.100	floor	YES	—
4.101	fmod	YES	—
4.102	fpclassify	YES	—
4.103	From_DATE_AND_TIME_VALUE	YES	—
4.104	From_TIME_VALUE	YES	—
4.105	fsetval	YES	—
4.106	ftoa	YES	—
4.107	fvar_value	YES	—
4.108	get_acknowledgement	NO	—
4.109	get_acknowledgement2	NO	—
4.110	get_block_instance_by_object_index	NO	—
4.111	get_block_instance_by_tag	NO	—
4.112	get_block_instance_count	NO	—
4.113	get_comm_error	NO	—
4.114	get_comm_error_string	NO	—
4.115	get_date	NO	—
4.116	get_date_lelem	NO	—
4.117	get_date_lelem2	NO	—
4.118	get_date_value	NO	—
4.119	get_date_value2	NO	—
4.120	GET_DD_REVISION	YES	—
4.121	get_dds_error	NO	—
4.122	GET_DEV_VAR_VALUE	YES	—
4.123	get_dev_var_value	YES	—
4.124	GET_DEVICE_REVISION	YES	—
4.125	GET_DEVICE_TYPE	YES	—
4.126	get_dictionary_string	YES	—
4.127	get_double	NO	—
4.128	get_double_lelem	NO	—

Reference	Builtin Name	Presence	Constraints
4.129	get_double_lelem2	NO	—
4.130	get_double_value	NO	—
4.131	get_double_value2	NO	—
4.132	get_enum_string	YES	—
4.133	get_float	NO	—
4.134	get_float_lelem	NO	—
4.135	get_float_lelem2	NO	—
4.136	get_float_value	NO	—
4.137	get_float_value2	NO	—
4.138	GET_LOCAL_VAR_VALUE	YES	—
4.139	get_local_var_value	YES	—
4.140	GET_MANUFACTURER	YES	—
4.141	get_more_status	NO	—
4.142	get_resolve_status	NO	—
4.143	get_response_code	NO	—
4.144	get_response_code_string	NO	—
4.145	get_rspcode_string	NO	—
4.146	get_rspcode_string_by_id	YES	—
4.147	get_signed	NO	—
4.148	get_signed_lelem	NO	—
4.149	get_signed_lelem2	NO	—
4.150	get_signed_value	NO	—
4.151	get_signed_value2	NO	—
4.152	get_status_code_string	NO	—
4.153	get_status_string	NO	—
4.154	get_stdlist_string	NO	—
4.155	get_string	NO	—
4.156	get_string_lelem	NO	—
4.157	get_string_lelem2	NO	—
4.158	get_string_value	NO	—
4.159	get_string_value2	NO	—
4.160	GET_TICK_COUNT	YES	—
4.161	Get_transfer_status	NO	—
4.162	get_unsigned	NO	—
4.163	get_unsigned_lelem	NO	—
4.164	get_unsigned_lelem2	NO	—
4.165	get_unsigned_value	NO	—
4.166	get_unsigned_value2	NO	—
4.167	get_variable_string	YES	—
4.168	GetCurrentDate	YES	—
4.169	GetCurrentDateAndTime	YES	—
4.170	GetCurrentTime	YES	—
4.171	iassign	YES	—
4.172	igetval	YES	—

Reference	Builtin Name	Presence	Constraints
4.173	IGNORE_ALL_COMM_STATUS	NO	—
4.174	IGNORE_ALL_DEVICE_STATUS	NO	—
4.175	IGNORE_ALL_RESPONSE_CODES	NO	—
4.176	IGNORE_COMM_ERROR	NO	—
4.177	IGNORE_COMM_STATUS	NO	—
4.178	IGNORE_DEVICE_STATUS	NO	—
4.179	IGNORE_NO_DEVICE	YES	—
4.180	IGNORE_RESPONSE_CODE	NO	—
4.181	int_value	YES	—
4.182	is_NaN	YES	—
4.183	isetval	YES	—
4.184	isOffline	YES	—
4.185	ITEM_ID	NO	—
4.186	itoa (version A)	YES	—
4.187	itoa (version B)	NO	—
4.188	ivar_value	YES	—
4.189	lassign	YES	—
4.190	lgetval	NO	—
4.191	ListDeleteElementAt	YES	—
4.192	ListDeleteElementAt2	NO	—
4.193	ListInsert	YES	—
4.194	ListInsert2	NO	—
4.195	log	YES	—
4.196	LOG_MESSAGE	YES	—
4.197	log10	YES	—
4.198	log2	YES	—
4.199	long_value	YES	—
4.200	LongToByte	YES	—
4.201	lsetval	YES	—
4.202	lvar_value	YES	—
4.203	Make_Time	YES	—
4.204	MEMBER_ID	NO	—
4.205	MenuDisplay	YES	—
4.206	method_abort	NO	—
4.207	nan	YES	—
4.208	NaN_value	YES	—
4.209	nanf	YES	—
4.210	ObjectReference	YES	—
4.211	openTransferPort	NO	—
4.212	pop_abort_method	YES	—
4.213	pow	YES	—
4.214	process_abort	YES	—
4.215	push_abort_method	YES	—
4.216	put_date	NO	—

Reference	Builtin Name	Presence	Constraints
4.217	put_date_value	NO	—
4.218	put_date_value2	NO	—
4.219	put_double	NO	—
4.220	put_double_value	NO	—
4.221	put_double_value2	NO	—
4.222	put_float	NO	—
4.223	put_float_value	NO	—
4.224	put_float_value2	NO	—
4.225	PUT_MESSAGE	YES	—
4.226	put_message	YES	—
4.227	put_signed	NO	—
4.228	put_signed_value	NO	—
4.229	put_signed_value2	NO	—
4.230	put_string	NO	—
4.231	put_string_value	NO	—
4.232	put_string_value2	NO	—
4.233	put_unsigned	NO	—
4.234	put_unsigned_value	NO	—
4.235	put_unsigned_value2	NO	—
4.236	re_read_file	NO	—
4.237	re_write_file	NO	—
4.238	read_value	NO	—
4.239	read_value2	NO	—
4.240	ReadCommand	YES	—
4.241	readItemFromDevice	NO	—
4.242	remove_abort_method (version A)	YES	—
4.243	remove_abort_method (version B)	NO	—
4.244	remove_all_abort_methods	YES	—
4.245	resolve_array_ref	NO	—
4.246	resolve_array_ref2	NO	—
4.247	resolve_block_ref	NO	—
4.248	resolve_block_ref2	NO	—
4.248	resolve_list_ref	NO	—
4.250	resolve_local_ref	NO	—
4.251	resolve_local_ref2	NO	—
4.252	resolve_param_list_ref	NO	—
4.253	resolve_param_ref	NO	—
4.254	resolve_param_ref2	NO	—
4.255	resolve_record_ref	NO	—
4.256	resolve_record_ref2	NO	—
4.257	ret_double_value	NO	—
4.258	ret_double_value2	NO	—
4.259	ret_float_value	NO	—
4.260	ret_float_value2	NO	—

Reference	Builtin Name	Presence	Constraints
4.261	ret_signed_value	NO	—
4.262	ret_signed_value2	NO	—
4.263	ret_unsigned_value	NO	—
4.264	ret_unsigned_value2	NO	—
4.265	retry_on_all_comm_errors	NO	—
4.266	RETRY_ON_ALL_COMM_STATUS	NO	—
4.267	RETRY_ON_ALL_DEVICE_STATUS	NO	—
4.268	RETRY_ON_ALL_RESPONSE_CODES	NO	—
4.269	retry_on_all_response_codes	NO	—
4.270	RETRY_ON_COMM_ERROR	NO	—
4.271	retry_on_comm_error	NO	—
4.272	RETRY_ON_COMM_STATUS	NO	—
4.273	RETRY_ON_DEVICE_STATUS	NO	—
4.274	RETRY_ON_NO_DEVICE	NO	—
4.275	RETRY_ON_RESPONSE_CODE	NO	—
4.276	retry_on_response_code	NO	—
4.277	round	YES	—
4.278	save_on_exit	YES	—
4.279	save_values	YES	—
4.280	seconds_to_TIME_VALUE	YES	—
4.281	seconds_to_TIME_VALUE8	YES	—
4.282	SELECT_FROM_LIST	YES	—
4.283	select_from_list	YES	—
4.284	select_from_menu	NO	—
4.285	select_from_menu2	NO	—
4.286	send	NO	—
4.287	send_all_values	NO	—
4.288	send_command	NO	—
4.289	send_command_trans	NO	—
4.290	send_on_exit	YES	—
4.291	send_trans	NO	—
4.292	send_value	NO	—
4.293	send_value2	NO	—
4.294	SET_NUMBER_OF_RETRIES	YES	—
4.295	sgetval	YES	—
4.296	ShortToByte	YES	—
4.297	sin	YES	—
4.298	sinh	YES	—
4.299	sqrt	YES	—
4.300	ssetval	YES	—
4.301	strcmp	YES	—
4.302	strleft	YES	—
4.303	strlen	YES	—
4.304	strlwr	YES	—

Reference	Builtin Name	Presence	Constraints
4.305	strmid	YES	—
4.306	strright	YES	—
4.307	strstr	YES	—
4.308	strtrim	YES	—
4.309	strupr	YES	—
4.310	tan	YES	—
4.311	tanh	YES	—
4.312	Time_To_Date	YES	—
4.313	TIME_VALUE_to_Hour	YES	—
4.314	TIME_VALUE_to_Minute	YES	—
4.315	TIME_VALUE_to_Second	YES	—
4.316	TIME_VALUE_to_seconds	YES	—
4.317	TIME_VALUE_to_string	YES	—
4.318	timet_to_string	YES	—
4.319	timet_to_TIME_VALUE	YES	—
4.320	timet_to_TIME_VALUE8	YES	—
4.321	To_Date	YES	—
4.322	To_Date_and_Time	YES	—
4.323	To_Time	YES	—
4.324	To_TIME_VALUE	YES	—
4.325	To_TIME_VALUE8	YES	—
4.326	trunc	YES	—
4.327	VARID	YES	—
4.328	vassign	YES	—
4.329	WriteCommand	YES	—
4.330	writeltemToDevice	NO	—
4.331	XMTR_ABORT_ON_ALL_COMM_STATUS	NO	—
4.332	XMTR_ABORT_ON_ALL_DATA	NO	—
4.333	XMTR_ABORT_ON_ALL_DEVICE_STATUS	NO	—
4.334	XMTR_ABORT_ON_ALL_RESPONSE_CODES	NO	—
4.335	XMTR_ABORT_ON_COMM_ERROR	NO	—
4.336	XMTR_ABORT_ON_COMM_STATUS	NO	—
4.337	XMTR_ABORT_ON_DATA	NO	—
4.338	XMTR_ABORT_ON_DEVICE_STATUS	NO	—
4.339	XMTR_ABORT_ON_NO_DEVICE	NO	—
4.340	XMTR_ABORT_ON_RESPONSE_CODE	NO	—
4.341	XMTR_IGNORE_ALL_COMM_STATUS	NO	—
4.342	XMTR_IGNORE_ALL_DATA	NO	—
4.343	XMTR_IGNORE_ALL_DEVICE_STATUS	NO	—
4.344	XMTR_IGNORE_ALL_RESPONSE_CODES	NO	—
4.345	XMTR_IGNORE_COMM_ERROR	NO	—
4.346	XMTR_IGNORE_COMM_STATUS	NO	—
4.347	XMTR_IGNORE_DATA	NO	—
4.348	XMTR_IGNORE_DEVICE_STATUS	NO	—

Reference	Builtin Name	Presence	Constraints
4.349	XMTR_IGNORE_NO_DEVICE	NO	—
4.350	XMTR_IGNORE_RESPONSE_CODE	NO	—
4.351	XMTR_RETRY_ON_ALL_COMM_STATUS	NO	—
4.352	XMTR_RETRY_ON_ALL_DATA	NO	—
4.353	XMTR_RETRY_ON_ALL_DEVICE_STATUS	NO	—
4.355	XMTR_RETRY_ON_ALL_RESPONSE_CODES	NO	—
4.356	XMTR_RETRY_ON_COMM_ERROR	NO	—
4.357	XMTR_RETRY_ON_COMM_STATUS	NO	—
4.358	XMTR_RETRY_ON_DATA	NO	—
4.359	XMTR_RETRY_ON_DEVICE_STATUS	NO	—
4.360	XMTR_RETRY_ON_NO_DEVICE	NO	—
4.361	XMTR_RETRY_ON_RESPONSE_CODE	NO	—

SOMMAIRE

AVANT-PROPOS.....	251
INTRODUCTION.....	253
1 Domaine d'application.....	254
2 Références normatives	254
3 Termes, définitions, acronymes et abréviations	254
3.1 Termes et définitions.....	254
3.2 Acronymes et abréviations	254
4 Bibliothèque de Builtin EDDL	255
4.1 Généralités	255
4.2 Conventions pour les descriptions de Builtin	255
4.3 Catégories de Builtin	256
4.3.1 Vue d'ensemble	256
4.3.2 Builtin de l'interface utilisateur.....	257
4.3.3 Builtin de communication.....	258
4.3.4 Builtin d'action	260
4.4 Builtin _ERROR	260
4.5 Builtin _TRACE	261
4.6 Builtin _WARNING	261
4.7 Builtin abort	261
4.8 Builtin abort_on_all_comm_errors.....	262
4.9 Builtin ABORT_ON_ALL_COMM_STATUS.....	262
4.10 Builtin ABORT_ON_ALL_DEVICE_STATUS.....	263
4.11 Builtin ABORT_ON_ALL_RESPONSE_CODES.....	263
4.12 Builtin abort_on_all_response_codes.....	264
4.13 Builtin ABORT_ON_COMM_ERROR.....	264
4.14 Builtin abort_on_comm_error.....	265
4.15 Builtin ABORT_ON_COMM_STATUS.....	265
4.16 Builtin ABORT_ON_DEVICE_STATUS.....	266
4.17 Builtin ABORT_ON_NO_DEVICE.....	266
4.18 Builtin ABORT_ON_RESPONSE_CODE.....	267
4.19 Builtin abort_on_response_code.....	268
4.20 Builtin abortTransferPort	269
4.21 Builtin abs.....	269
4.22 Builtin ACKNOWLEDGE	269
4.23 Builtin acknowledge	270
4.24 Builtin acos	270
4.25 Builtin add_abort_method (version A)	270
4.26 Builtin add_abort_method (version B)	271
4.27 Builtin AddTime.....	271
4.28 Builtin asin	272
4.29 Builtin assign	272
4.30 Builtin assign_double	273
4.31 Builtin assign_float.....	273
4.32 Builtin assign_int.....	273
4.33 Builtin assign_var.....	274
4.34 Builtin assign2	274

4.35	Builtin atan.....	275
4.36	Builtin atof	275
4.37	Builtin atoi.....	276
4.38	Builtin browselidentity	276
4.39	Builtin BUILD_MESSAGE	276
4.40	Builtin ByteToDouble.....	277
4.41	Builtin ByteToFloat.....	277
4.42	Builtin ByteToLong	278
4.43	Builtin ByteToShort	278
4.44	Builtin cbrt	279
4.45	Builtin ceil	279
4.46	Builtin closeTransferPort	279
4.47	Builtin cos	280
4.48	Builtin cosh	280
4.49	Builtin dassign	280
4.50	Builtin DATE_AND_TIME_VALUE_to_string.....	281
4.51	Builtin Date_to_DayOfMonth.....	281
4.52	Builtin DATE_to_days.....	282
4.53	Builtin Date_to_Month	282
4.54	Builtin DATE_to_string	282
4.55	Builtin Date_To_Time.....	283
4.56	Builtin Date_to_Year	283
4.57	Builtin days_to_DATE.....	283
4.58	Builtin DELAY	284
4.59	Builtin delay	284
4.60	Builtin DELAY_TIME	285
4.61	Builtin delayfor	285
4.62	Builtin delayfor2	286
4.63	Builtin DICT_ID	287
4.64	Builtin dictionnaire_string	287
4.65	Builtin DiffTime	287
4.66	Builtin discard_on_exit	288
4.67	Builtin DISPLAY	288
4.68	Builtin display	289
4.69	Builtin display_bitenum.....	289
4.70	Builtin display_builtin_error	290
4.71	Builtin display_comm_error	290
4.72	Builtin display_comm_status.....	291
4.73	Builtin display_device_status.....	291
4.74	Builtin display_dynamics	292
4.75	Builtin display_dynamics2	292
4.76	Builtin display_message	293
4.77	Builtin display_message2	294
4.78	Builtin display_response_code	294
4.79	Builtin display_response_status.....	295
4.80	Builtin display_xmtr_status	295
4.81	Builtin DoubleToByte.....	296
4.82	Builtin drand.....	296
4.83	Builtin dseed	297

4.84	Builtin edit_device_value	297
4.85	Builtin edit_device_value2	298
4.86	Builtin edit_local_value	299
4.87	Builtin edit_local_value2	300
4.88	Builtin exp	301
4.89	Builtin ext_send_command	301
4.90	Builtin ext_send_command_trans	302
4.91	Builtin fail_on_all_comm_errors	303
4.92	Builtin fail_on_all_response_codes	303
4.93	Builtin fail_on_comm_error	304
4.94	Builtin fail_on_response_code	304
4.95	Builtin fassign	305
4.96	Builtin fGetByte	305
4.97	Builtin fgetval	306
4.98	Builtin float_value	306
4.99	Builtin FloatToByte	306
4.100	Builtin floor	307
4.101	Builtin fmod	307
4.102	Builtin fpclassify	308
4.103	Builtin From_DATE_AND_TIME_VALUE	308
4.104	Builtin From_TIME_VALUE	309
4.105	Builtin fsetval	309
4.106	Builtin ftoa	310
4.107	Builtin fvar_value	310
4.108	Builtin get_acknowledgement	310
4.109	Builtin get_acknowledgement2	311
4.110	Builtin get_block_instance_by_objet_index	311
4.111	Builtin get_block_instance_by_tag	312
4.112	Builtin get_block_instance_count	312
4.113	Builtin get_comm_error	313
4.114	Builtin get_comm_error_string	314
4.115	Builtin get_date	314
4.116	Builtin get_date_lelem	315
4.117	Builtin get_date_lelem2	315
4.118	Builtin get_date_value	316
4.119	Builtin get_date_value2	316
4.120	Builtin GET_DD_REVISION	317
4.121	Builtin get_dds_error	317
4.122	Builtin GET_DEV_VAR_VALUE	318
4.123	Builtin get_dev_var_value	319
4.124	Builtin GET_DEVICE_REVISION	319
4.125	Builtin GET_DEVICE_TYPE	319
4.126	Builtin get_dictionary_string	320
4.127	Builtin get_double	320
4.128	Builtin get_double_lelem	321
4.129	Builtin get_double_lelem2	321
4.130	Builtin get_double_value	322
4.131	Builtin get_double_value2	322
4.132	Builtin get_enum_string	323

4.133	Builtin get_float	323
4.134	Builtin get_float_lelem	324
4.135	Builtin get_float_lelem2	324
4.136	Builtin get_float_value	325
4.137	Builtin get_float_value2	325
4.138	Builtin GET_LOCAL_VAR_VALUE	326
4.139	Builtin get_local_var_value	326
4.140	Builtin GET_MANUFACTURER	327
4.141	Builtin get_more_status	327
4.142	Builtin get_resolve_status	328
4.143	Builtin get_response_code	328
4.144	Builtin get_response_code_string	329
4.145	Builtin get_rspcode_string	330
4.146	Builtin get_rspcode_string_by_id	330
4.147	Builtin get_signed	331
4.148	Builtin get_signed_lelem	331
4.149	Builtin get_signed_lelem2	332
4.150	Builtin get_signed_value	332
4.151	Builtin get_signed_value2	333
4.152	Builtin get_status_code_string	333
4.153	Builtin get_status_string	334
4.154	Builtin get_stddict_string	334
4.155	Builtin get_string	335
4.156	Builtin get_string_lelem	336
4.157	Builtin get_string_lelem2	336
4.158	Builtin get_string_value	337
4.159	Builtin get_string_value2	337
4.160	Builtin GET_TICK_COUNT	338
4.161	Builtin get_transfer_status	338
4.162	Builtin get_unsigned	339
4.163	Builtin get_unsigned_lelem	339
4.164	Builtin get_unsigned_lelem2	340
4.165	Builtin get_unsigned_value	340
4.166	Builtin get_unsigned_value2	341
4.167	Builtin get_variable_string	342
4.168	Builtin GetCurrentDate	342
4.169	Builtin GetCurrentDateAndTime	343
4.170	Builtin GetCurrentTime	343
4.171	Builtin iassign	343
4.172	Builtin igetval	344
4.173	Builtin IGNORE_ALL_COMM_STATUS	344
4.174	Builtin IGNORE_ALL_DEVICE_STATUS	344
4.175	Builtin IGNORE_ALL_RESPONSE_CODES	345
4.176	Builtin IGNORE_COMM_ERROR	346
4.177	Builtin IGNORE_COMM_STATUS	346
4.178	Builtin IGNORE_DEVICE_STATUS	347
4.179	Builtin IGNORE_NO_DEVICE	347
4.180	Builtin IGNORE_RESPONSE_CODE	348
4.181	Builtin int_value	348

4.182	Builtin is_NaN	349
4.183	Builtin isetval	349
4.184	Builtin isOffline	349
4.185	Builtin ITEM_ID	350
4.186	Builtin itoa (version A)	350
4.187	Builtin itoa (version B)	350
4.188	Builtin ivar_value	351
4.189	Builtin lassign	351
4.190	Builtin lgetval	352
4.191	Builtin ListDeleteElementAt	352
4.192	Builtin ListDeleteElementAt2	353
4.193	Builtin ListInsert	353
4.194	Builtin ListInsert2	354
4.195	Builtin log	354
4.196	Builtin LOG_MESSAGE	355
4.197	Builtin log10	355
4.198	Builtin log2	355
4.199	Builtin long_value	356
4.200	Builtin LongToByte	356
4.201	Builtin lsetval	357
4.202	Builtin lvar_value	357
4.203	Builtin Make_Time	357
4.204	Builtin MEMBER_ID	358
4.205	Builtin MenuDisplay	358
4.206	Builtin method_abort	359
4.207	Builtin nan	359
4.208	Builtin NaN_value	360
4.209	Builtin nanf	360
4.210	Builtin ObjectReference	361
4.211	Builtin openTransferPort	361
4.212	Builtin pop_abort_method	362
4.213	Builtin pow	362
4.214	Builtin process_abort	362
4.215	Builtin push_abort_method	363
4.216	Builtin put_date	363
4.217	Builtin put_date_value	364
4.218	Builtin put_date_value2	364
4.219	Builtin put_double	365
4.220	Builtin put_double_value	366
4.221	Builtin put_double_value2	366
4.222	Builtin put_float	367
4.223	Builtin put_float_value	367
4.224	Builtin put_float_value2	368
4.225	Builtin PUT_MESSAGE	369
4.226	Builtin put_message	369
4.227	Builtin put_signed	370
4.228	Builtin put_signed_value	371
4.229	Builtin put_signed_value2	371
4.230	Builtin put_string	372

4.231	Builtin put_string_value	373
4.232	Builtin put_string_value2	373
4.233	Builtin put_unsigned	374
4.234	Builtin put_unsigned_value	375
4.235	Builtin put_unsigned_value2	375
4.236	Builtin re_read_file	376
4.237	Builtin re_write_file	377
4.238	Builtin read_value	377
4.239	Builtin read_value2	377
4.240	Builtin ReadCommand	378
4.241	Builtin readItemFromDevice	378
4.242	Builtin remove_abort_method (version A)	379
4.243	Builtin remove_abort_method (version B)	379
4.244	Builtin remove_all_abort_methods	380
4.245	Builtin resolve_array_ref	380
4.246	Builtin resolve_array_ref2	381
4.247	Builtin resolve_block_ref	381
4.248	Builtin resolve_block_ref2	382
4.249	Builtin resolve_list_ref	383
4.250	Builtin resolve_local_ref	383
4.251	Builtin resolve_local_ref2	384
4.252	Builtin resolve_param_list_ref	384
4.253	Builtin resolve_param_ref	385
4.254	Builtin resolve_param_ref2	385
4.255	Builtin resolve_record_ref	386
4.256	Builtin resolve_record_ref2	386
4.257	Builtin ret_double_value	387
4.258	Builtin ret_double_value2	388
4.259	Builtin ret_float_value	388
4.260	Builtin ret_float_value2	388
4.261	Builtin ret_signed_value	389
4.262	Builtin ret_signed_value2	389
4.263	Builtin ret_unsigned_value	390
4.264	Builtin ret_unsigned_value2	390
4.265	Builtin retry_on_all_comm_errors	390
4.266	Builtin RETRY_ON_ALL_COMM_STATUS	391
4.267	Builtin RETRY_ON_ALL_DEVICE_STATUS	391
4.268	Builtin RETRY_ON_ALL_RESPONSE_CODES	392
4.269	Builtin retry_on_all_response_codes	392
4.270	Builtin RETRY_ON_COMM_ERROR	393
4.271	Builtin retry_on_comm_error	393
4.272	Builtin RETRY_ON_COMM_STATUS	394
4.273	Builtin RETRY_ON_DEVICE_STATUS	394
4.274	Builtin RETRY_ON_NO_DEVICE	395
4.275	Builtin RETRY_ON_RESPONSE_CODE	396
4.276	Builtin retry_on_response_code	396
4.277	Builtin round	397
4.278	Builtin save_on_exit	397
4.279	Builtin save_values	398

4.280	Builtin seconds_to_TIME_VALUE	398
4.281	Builtin seconds_to_TIME_VALUE8	398
4.282	Builtin SELECT_FROM_LIST	399
4.283	Builtin select_from_list	399
4.284	Builtin select_from_menu	400
4.285	Builtin select_from_menu2	401
4.286	Builtin send	402
4.287	Builtin send_all_values	403
4.288	Builtin send_command	403
4.289	Builtin send_command_trans	404
4.290	Builtin send_on_exit	404
4.291	Builtin send_trans	405
4.292	Builtin send_value	405
4.293	Builtin send_value2	406
4.294	Builtin SET_NUMBER_OF_RETRIES	407
4.295	Builtin sgetval	407
4.296	Builtin ShortToByte	407
4.297	Builtin sin	408
4.298	Builtin sinh	408
4.299	Builtin sqrt	408
4.300	Builtin ssetval	409
4.301	Builtin strcmp	409
4.302	Builtin strleft	409
4.303	Builtin strlen	410
4.304	Builtin strlwr	410
4.305	Builtin strmid	410
4.306	Builtin strright	411
4.307	Builtin strstr	411
4.308	Builtin strtrim	411
4.309	Builtin strupr	412
4.310	Builtin tan	412
4.311	Builtin tanh	412
4.312	Builtin Time_To_Date	413
4.313	Builtin TIME_VALUE_to_Hour	413
4.314	Builtin TIME_VALUE_to_Minute	413
4.315	Builtin TIME_VALUE_to_Second	414
4.316	Builtin TIME_VALUE_to_seconds	414
4.317	Builtin TIME_VALUE_to_string	415
4.318	Builtin timet_to_string	415
4.319	Builtin timet_to_TIME_VALUE	416
4.320	Builtin timet_to_TIME_VALUE8	416
4.321	Builtin To_Date	416
4.322	Builtin To_Date_and_Time	417
4.323	Builtin To_Time	417
4.324	Builtin To_TIME_VALUE	418
4.325	Builtin To_TIME_VALUE8	418
4.326	Builtin trunc	419
4.327	Builtin VARID	419
4.328	Builtin vassign	419

4.329	Builtin WriteCommand	420
4.330	Builtin writeltemToDevice	420
4.331	Builtin XMTR_ABORT_ON_ALL_COMM_STATUS	421
4.332	Builtin XMTR_ABORT_ON_ALL_DATA	421
4.333	Builtin XMTR_ABORT_ON_ALL_DEVICE_STATUS	422
4.334	Builtin XMTR_ABORT_ON_ALL_RESPONSE_CODES	422
4.335	Builtin XMTR_ABORT_ON_COMM_ERROR	423
4.336	Builtin XMTR_ABORT_ON_COMM_STATUS	423
4.337	Builtin XMTR_ABORT_ON_DATA	424
4.338	Builtin XMTR_ABORT_ON_DEVICE_STATUS	424
4.339	Builtin XMTR_ABORT_ON_NO_DEVICE	425
4.340	Builtin XMTR_ABORT_ON_RESPONSE_CODE	425
4.341	Builtin XMTR_IGNORE_ALL_COMM_STATUS	426
4.342	Builtin XMTR_IGNORE_ALL_DATA	427
4.343	Builtin XMTR_IGNORE_ALL_DEVICE_STATUS	427
4.344	Builtin XMTR_IGNORE_ALL_RESPONSE_CODES	428
4.345	Builtin XMTR_IGNORE_COMM_ERROR	428
4.346	Builtin XMTR_IGNORE_COMM_STATUS	429
4.347	Builtin XMTR_IGNORE_DATA	429
4.348	Builtin XMTR_IGNORE_DEVICE_STATUS	430
4.349	Builtin XMTR_IGNORE_NO_DEVICE	430
4.350	Builtin XMTR_IGNORE_RESPONSE_CODE	431
4.351	Builtin XMTR_RETRY_ON_ALL_COMM_STATUS	431
4.352	Builtin XMTR_RETRY_ON_ALL_DATA	432
4.353	Builtin XMTR_RETRY_ON_ALL_DEVICE_STATUS	432
4.354	Builtin XMTR_RETRY_ON_ALL_RESPONSE_CODE (supprimé)	433
4.355	Builtin XMTR_RETRY_ON_ALL_RESPONSE_CODES	433
4.356	Builtin XMTR_RETRY_ON_COMM_ERROR	434
4.357	Builtin XMTR_RETRY_ON_COMM_STATUS	434
4.358	Builtin XMTR_RETRY_ON_DATA	435
4.359	Builtin XMTR_RETRY_ON_DEVICE_STATUS	435
4.360	Builtin XMTR_RETRY_ON_NO_DEVICE	436
4.361	Builtin XMTR_RETRY_ON_RESPONSE_CODE	436
5	Codes de retour de Builtin	437
Annexe A	(normative) Profils des Builtins	439
A.1	Conventions pour les profils de Builtin	439
A.2	Profils pour PROFIBUS et PROFINET	439
A.3	Profils pour FOUNDATION™ fieldbus	447
A.4	Profils pour HART®	456
A.5	Profils pour les serveurs de communication	464
Tableau 1	– Format pour les tableaux d'élément lexical de Builtin	255
Tableau 2	– Contenu du Tableau d'élément lexical	256
Tableau 3	– Usage des Builtin	256
Tableau 4	– Builtin de l'interface utilisateur	257
Tableau 5	– Builtin de communication	258
Tableau 6	– Builtin d'action	260
Tableau 7	– Builtin_ERROR	260

Tableau 8 – Builtin _TRACE.....	261
Tableau 9 – Builtin _WARNING.....	261
Tableau 10 – Builtin abort.....	262
Tableau 11 – Builtin abort_on_all_comm_errors.....	262
Tableau 12 – Builtin ABORT_ON_ALL_COMM_STATUS.....	263
Tableau 13 – Builtin ABORT_ON_ALL_DEVICE_STATUS.....	263
Tableau 14 – Builtin ABORT_ON_ALL_RESPONSE_CODES.....	264
Tableau 15 – Builtin abort_on_all_response_codes.....	264
Tableau 16 – Builtin ABORT_ON_COMM_ERROR.....	265
Tableau 17 – Builtin abort_on_comm_error.....	265
Tableau 18 – Builtin ABORT_ON_COMM_STATUS.....	266
Tableau 19 – Builtin ABORT_ON_DEVICE_STATUS.....	266
Tableau 20 – Builtin ABORT_ON_NO_DEVICE.....	267
Tableau 21 – Builtin ABORT_ON_RESPONSE_CODE.....	268
Tableau 22 – Builtin abort_on_response_code.....	268
Tableau 23 – Builtin abortTransferPort.....	269
Tableau 24 – Builtin abs.....	269
Tableau 25 – Builtin ACKNOWLEDGE.....	269
Tableau 26 – Builtin acknowledge.....	270
Tableau 27 – Builtin acos.....	270
Tableau 28 – Builtin add_abort_method.....	271
Tableau 29 – Builtin add_abort_method.....	271
Tableau 30 – Builtin AddTime.....	272
Tableau 31 – Builtin asin.....	272
Tableau 32 – Builtin assign.....	273
Tableau 33 – Builtin assign_double.....	273
Tableau 34 – Builtin assign_float.....	273
Tableau 35 – Builtin assign_int.....	274
Tableau 36 – Builtin assign_var.....	274
Tableau 37 – Builtin assign2.....	275
Tableau 38 – Builtin atan.....	275
Tableau 39 – Builtin atof.....	276
Tableau 40 – Builtin atoi.....	276
Tableau 41 – Builtin browseldentity.....	276
Tableau 42 – Builtin BUILD_MESSAGE.....	277
Tableau 43 – Builtin ByteToDouble.....	277
Tableau 44 – Builtin ByteToFloat.....	278
Tableau 45 – Builtin ByteToLong.....	278
Tableau 46 – Builtin ByteToShort.....	279
Tableau 47 – Builtin cbrt.....	279
Tableau 48 – Builtin ceil.....	279
Tableau 49 – Builtin closeTransferPort.....	280
Tableau 50 – Builtin cos.....	280

Tableau 51 – Builtin cosh	280
Tableau 52 – Builtin dassign	281
Tableau 53 – Builtin DATE_AND_TIME_VALUE_to_string	281
Tableau 54 – Builtin Date_to_DayOfMonth	281
Tableau 55 – Builtin DATE_to_days	282
Tableau 56 – Builtin Date_to_Month	282
Tableau 57 – Builtin DATE_to_string	283
Tableau 58 – Builtin Date_To_Time	283
Tableau 59 – Builtin Date_to_Year	283
Tableau 60 – Builtin days_to_DATE	284
Tableau 61 – Builtin DELAY	284
Tableau 62 – Builtin delay	285
Tableau 63 – Builtin DELAY_TIME	285
Tableau 64 – Builtin delayfor	286
Tableau 65 – Builtin delayfor2	286
Tableau 66 – Builtin DICT_ID	287
Tableau 67 – Builtin dictionary_string	287
Tableau 68 – Builtin DiffTime	288
Tableau 69 – Builtin discard_on_exit	288
Tableau 70 – Builtin DISPLAY	289
Tableau 71 – Builtin display	289
Tableau 72 – Builtin display_bitenum	290
Tableau 73 – Builtin display_builtin_error	290
Tableau 74 – Builtin display_comm_error	291
Tableau 75 – Builtin display_comm_status	291
Tableau 76 – Builtin display_device_status	291
Tableau 77 – Builtin display_dynamics	292
Tableau 78 – Builtin display_dynamics2	293
Tableau 79 – Builtin display_message	293
Tableau 80 – Builtin display_message2	294
Tableau 81 – Builtin display_response_code	295
Tableau 82 – Builtin display_response_status	295
Tableau 83 – Builtin display_xmtr_status	296
Tableau 84 – Builtin DoubleToByte	296
Tableau 85 – Builtin drand	297
Tableau 86 – Builtin dseed	297
Tableau 87 – Builtin edit_device_value	298
Tableau 88 – Builtin edit_device_value2	299
Tableau 89 – Builtin edit_local_value	300
Tableau 90 – Builtin edit_local_value2	301
Tableau 91 – Builtin exp	301
Tableau 92 – Builtin ext_send_command	302
Tableau 93 – Builtin ext_send_command_trans	302

Tableau 94 – Builtin fail_on_all_comm_errors.....	303
Tableau 95 – Builtin fail_on_all_response_codes.....	304
Tableau 96 – Builtin fail_on_comm_error.....	304
Tableau 97 – Builtin fail_on_response_code.....	305
Tableau 98 – Builtin fassign.....	305
Tableau 99 – Builtin fGetByte.....	306
Tableau 100 – Builtin fgetval.....	306
Tableau 101 – Builtin float_value.....	306
Tableau 102 – Builtin FloatToByte.....	307
Tableau 103 – Builtin floor.....	307
Tableau 104 – Builtin fmod.....	308
Tableau 105 – Builtin fpclassifyd.....	308
Tableau 106 – Builtin From_DATE_AND_TIME_VALUE.....	309
Tableau 107 – Builtin From_TIME_VALUE.....	309
Tableau 108 – Builtin fsetval.....	309
Tableau 109 – Builtin ftoa.....	310
Tableau 110 – Builtin fvar_value.....	310
Tableau 111 – Builtin get_acknowledgement.....	311
Tableau 112 – Builtin get_acknowledgement2.....	311
Tableau 113 – Builtin get_block_instance_by_object_index.....	312
Tableau 114 – Builtin get_block_instance_by_tag.....	312
Tableau 115 – Builtin get_block_instance_count.....	313
Tableau 116 – Builtin get_comm_error.....	313
Tableau 117 – Builtin get_comm_error_string.....	314
Tableau 118 – Builtin get_date.....	314
Tableau 119 – Builtin get_date_lelem.....	315
Tableau 120 – Builtin get_date_lelem2.....	316
Tableau 121 – Builtin get_date_value.....	316
Tableau 122 – Builtin get_date_value2.....	317
Tableau 123 – Builtin GET_DD_REVISION.....	317
Tableau 124 – Builtin get_dds_error.....	318
Tableau 125 – Builtin GET_DEV_VAR_VALUE.....	318
Tableau 126 – Builtin get_dev_var_value.....	319
Tableau 127 – Builtin GET_DEVICE_REVISION.....	319
Tableau 128 – Builtin GET_DEVICE_TYPE.....	320
Tableau 129 – Builtin get_dictionary_string.....	320
Tableau 130 – Builtin get_double.....	321
Tableau 131 – Builtin get_double_lelem.....	321
Tableau 132 – Builtin get_double_lelem2.....	322
Tableau 133 – Builtin get_double_value.....	322
Tableau 134 – Builtin get_double_value2.....	323
Tableau 135 – Builtin get_enum_string.....	323
Tableau 136 – Builtin get_float.....	324

Tableau 137 – Builtin get_float_lelem	324
Tableau 138 – Builtin get_float_lelem2	325
Tableau 139 – Builtin get_float_value	325
Tableau 140 – Builtin get_float_value2	326
Tableau 141 – Builtin GET_LOCAL_VAR_VALUE	326
Tableau 142 – Builtin get_local_var_value	327
Tableau 143 – Builtin GET_MANUFACTURER	327
Tableau 144 – Builtin get_more_status	328
Tableau 145 – Builtin get_resolve_status	328
Tableau 146 – Builtin get_response_code	329
Tableau 147 – Builtin get_response_code_string	330
Tableau 148 – Builtin get_rspcode_string	330
Tableau 149 – Builtin get_rspcode_string_by_id	331
Tableau 150 – Builtin get_signed	331
Tableau 151 – Builtin get_signed_lelem	332
Tableau 152 – Builtin get_signed_lelem2	332
Tableau 153 – Builtin get_signed_value	333
Tableau 154 – Builtin get_signed_value2	333
Tableau 155 – Builtin get_status_code_string	334
Tableau 156 – Builtin get_status_string	334
Tableau 157 – Builtin get_stdidct_string	335
Tableau 158 – Builtin get_string	335
Tableau 159 – Builtin get_string_lelem	336
Tableau 160 – Builtin get_string_lelem2	337
Tableau 161 – Builtin get_string_value	337
Tableau 162 – Builtin get_string_value2	338
Tableau 163 – Builtin GET_TICK_COUNT	338
Tableau 164 – Builtin get_transfer_status	339
Tableau 165 – Builtin get_unsigned	339
Tableau 166 – Builtin get_unsigned_lelem	340
Tableau 167 – Builtin get_unsigned_lelem2	340
Tableau 168 – Builtin get_unsigned_value	341
Tableau 169 – Builtin get_unsigned_value2	342
Tableau 170 – Builtin get_variable_string	342
Tableau 171 – Builtin GetCurrentDate	342
Tableau 172 – Builtin GetCurrentDateAndTime	343
Tableau 173 – Builtin GetCurrentTime	343
Tableau 174 – Builtin iassign	343
Tableau 175 – Builtin igetval	344
Tableau 176 – Builtin IGNORE_ALL_COMM_STATUS	344
Tableau 177 – Builtin IGNORE_ALL_DEVICE_STATUS	345
Tableau 178 – Builtin IGNORE_ALL_RESPONSE_CODES	345
Tableau 179 – Builtin IGNORE_COMM_ERROR	346

Tableau 180 – Builtin IGNORE_COMM_STATUS.....	347
Tableau 181 – Builtin IGNORE_DEVICE_STATUS.....	347
Tableau 182 – Builtin IGNORE_NO_DEVICE.....	348
Tableau 183 – Builtin IGNORE_RESPONSE_CODE.....	348
Tableau 184 – Builtin int_value.....	349
Tableau 185 – Builtin is_NaN.....	349
Tableau 186 – Builtin isetval.....	349
Tableau 187 – Builtin isOffline.....	350
Tableau 188 – Builtin ITEM_ID.....	350
Tableau 189 – Builtin itoa (version A).....	350
Tableau 190 – Builtin itoa (version B).....	351
Tableau 191 – Builtin ivar_value.....	351
Tableau 192 – Builtin lassign.....	351
Tableau 193 – Builtin lgetval.....	352
Tableau 194 – Builtin ListDeleteElementAt.....	352
Tableau 195 – Builtin ListDeleteElementAt2.....	353
Tableau 196 – Builtin ListInsert.....	354
Tableau 197 – Builtin ListInsert2.....	354
Tableau 198 – Builtin log.....	355
Tableau 199 – Builtin LOG_MESSAGE.....	355
Tableau 200 – Builtin log10.....	355
Tableau 201 – Builtin log2.....	356
Tableau 202 – Builtin long_value.....	356
Tableau 203 – Builtin LongToByte.....	356
Tableau 204 – Builtin lsetval.....	357
Tableau 205 – Builtin lvar_value.....	357
Tableau 206 – Builtin Make_Time.....	358
Tableau 207 – Builtin MEMBER_ID.....	358
Tableau 208 – Builtin MenuDisplay.....	359
Tableau 209 – Builtin method_abort.....	359
Tableau 210 – Builtin nan.....	360
Tableau 211 – Builtin NaN_value.....	360
Tableau 212 – Builtin nanf.....	361
Tableau 213 – Builtin ObjectReference.....	361
Tableau 214 – Builtin openTransferPort.....	362
Tableau 215 – Builtin pop_abort_method.....	362
Tableau 216 – Builtin pow.....	362
Tableau 217 – Builtin process_abort.....	363
Tableau 218 – Builtin push_abort_method.....	363
Tableau 219 – Builtin put_date.....	363
Tableau 220 – Builtin put_date_value.....	364
Tableau 221 – Builtin put_date_value2.....	365
Tableau 222 – Builtin put_double.....	365

Tableau 223 – Builtin put_double_value	366
Tableau 224 – Builtin put_double_value2	367
Tableau 225 – Builtin put_float	367
Tableau 226 – Builtin put_float_value	368
Tableau 227 – Builtin put_float_value2	368
Tableau 228 – Options de format	369
Tableau 229 – Builtin PUT_MESSAGE	369
Tableau 230 – Builtin put_message	370
Tableau 231 – Builtin put_signed	370
Tableau 232 – Builtin put_signed_value	371
Tableau 233 – Builtin put_signed_value2	372
Tableau 234 – Builtin put_string	372
Tableau 235 – Builtin put_string_value	373
Tableau 236 – Builtin put_string_value2	374
Tableau 237 – Builtin put_unsigned	375
Tableau 238 – Builtin put_unsigned_value	375
Tableau 239 – Builtin put_unsigned_value2	376
Tableau 240 – Builtin re_read_file	376
Tableau 241 – Builtin re_write_file	377
Tableau 242 – Builtin read_value	377
Tableau 243 – Builtin read_value2	378
Tableau 244 – Builtin ReadCommand	378
Tableau 245 – Builtin readItemFromDevice	379
Tableau 246 – Builtin remove_abort_method	379
Tableau 247 – Builtin remove_abort_method	380
Tableau 248 – Builtin remove_all_abort_methods	380
Tableau 249 – Builtin resolve_array_ref	381
Tableau 250 – Builtin resolve_array_ref2	381
Tableau 251 – Builtin resolve_block_ref	382
Tableau 252 – Builtin resolve_block_ref2	383
Tableau 253 – Builtin resolve_list_ref	383
Tableau 254 – Builtin resolve_local_ref	384
Tableau 255 – Builtin resolve_local_ref2	384
Tableau 256 – Builtin resolve_param_list_ref	385
Tableau 257 – Builtin resolve_param_ref	385
Tableau 258 – Builtin resolve_param_ref2	386
Tableau 259 – Builtin resolve_record_ref	386
Tableau 260 – Builtin resolve_record_ref2	387
Tableau 261 – Builtin ret_double_value	387
Tableau 262 – Builtin ret_double_value2	388
Tableau 263 – Builtin ret_float_value	388
Tableau 264 – Builtin ret_float_value2	389
Tableau 265 – Builtin ret_signed_value	389

Tableau 266 – Builtin ret_signed_value2	389
Tableau 267 – Builtin ret_unsigned_value	390
Tableau 268 – Builtin ret_unsigned_value2	390
Tableau 269 – Builtin retry_on_all_comm_errors	391
Tableau 270 – Builtin RETRY_ON_ALL_COMM_STATUS	391
Tableau 271 – Builtin RETRY_ON_ALL_DEVICE_STATUS	392
Tableau 272 – Builtin RETRY_ON_ALL_RESPONSE_CODES	392
Tableau 273 – Builtin retry_on_all_response_codes	393
Tableau 274 – Builtin RETRY_ON_COMM_ERROR	393
Tableau 275 – Builtin retry_on_comm_error	394
Tableau 276 – Builtin RETRY_ON_COMM_STATUS	394
Tableau 277 – Builtin RETRY_ON_DEVICE_STATUS	395
Tableau 278 – Builtin RETRY_ON_NO_DEVICE	395
Tableau 279 – Builtin RETRY_ON_RESPONSE_CODE	396
Tableau 280 – Builtin retry_on_response_code	397
Tableau 281 – Builtin round	397
Tableau 282 – Builtin save_on_exit	398
Tableau 283 – Builtin save_values	398
Tableau 284 – Builtin seconds_to_TIME_VALUE	398
Tableau 285 – Builtin seconds_to_TIME_VALUE	399
Tableau 286 – Builtin SELECT_FROM_LIST	399
Tableau 287 – Builtin select_from_list	400
Tableau 288 – Builtin select_from_menu	401
Tableau 289 – Builtin select_from_menu2	402
Tableau 290 – Builtin send	402
Tableau 291 – Builtin send_all_values	403
Tableau 292 – Builtin send_command	403
Tableau 293 – Builtin send_command_trans	404
Tableau 294 – Builtin send_on_exit	405
Tableau 295 – Builtin send_trans	405
Tableau 296 – Builtin send_value	406
Tableau 297 – Builtin send_value2	406
Tableau 298 – Builtin SET_NUMBER_OF_RETRIES	407
Tableau 299 – Builtin sgetval	407
Tableau 300 – Builtin ShortToByte	408
Tableau 301 – Builtin sin	408
Tableau 302 – Builtin sinh	408
Tableau 303 – Builtin sqrt	409
Tableau 304 – Builtin ssetval	409
Tableau 305 – Builtin strcmp	409
Tableau 306 – Builtin strleft	410
Tableau 307 – Builtin strlen	410
Tableau 308 – Builtin strlwr	410

Tableau 309 – Builtin strmid	411
Tableau 310 – Builtin strright	411
Tableau 311 – Builtin strstr	411
Tableau 312 – Builtin strtrim	412
Tableau 313 – Builtin strupr	412
Tableau 314 – Builtin tan	412
Tableau 315 – Builtin tanh	413
Tableau 316 – Builtin Time_To_Date	413
Tableau 317 – Builtin TIME_VALUE_to_Hour	413
Tableau 318 – Builtin TIME_VALUE_to_Minute	414
Tableau 319 – Builtin TIME_VALUE_to_Second	414
Tableau 320 – Builtin TIME_VALUE_to_seconds	415
Tableau 321 – Builtin TIME_VALUE_to_string	415
Tableau 322 – Builtin timet_to_string	416
Tableau 323 – Builtin timet_to_TIME_VALUE	416
Tableau 324 – Builtin timet_to_TIME_VALUE8	416
Tableau 325 – Builtin To_Date	417
Tableau 326 – Builtin To_Date_and_Time	417
Tableau 327 – Builtin To_Time	418
Tableau 328 – Builtin To_TIME_VALUE	418
Tableau 329 – Builtin To_TIME_VALUE8	419
Tableau 330 – Builtin trunc	419
Tableau 331 – Builtin VARID	419
Tableau 332 – Builtin vassign	420
Tableau 333 – Builtin WriteCommand	420
Tableau 334 – Builtin writeItemToDevice	420
Tableau 335 – Builtin XMTR_ABORT_ON_ALL_COMM_STATUS	421
Tableau 336 – Builtin XMTR_ABORT_ON_ALL_DATA	421
Tableau 337 – Builtin XMTR_ABORT_ON_ALL_DEVICE_STATUS	422
Tableau 338 – Builtin XMTR_ABORT_ON_ALL_RESPONSE_CODES	423
Tableau 339 – Builtin XMTR_ABORT_ON_COMM_ERROR	423
Tableau 340 – Builtin XMTR_ABORT_ON_COMM_STATUS	424
Tableau 341 – Builtin XMTR_ABORT_ON_DATA	424
Tableau 342 – Builtin XMTR_ABORT_ON_DEVICE_STATUS	425
Tableau 343 – Builtin XMTR_ABORT_ON_NO_DEVICE	425
Tableau 344 – Builtin XMTR_ABORT_ON_RESPONSE_CODE	426
Tableau 345 – Builtin XMTR_IGNORE_ALL_COMM_STATUS	426
Tableau 346 – Builtin XMTR_IGNORE_ALL_DATA	427
Tableau 347 – Builtin XMTR_IGNORE_ALL_DEVICE_STATUS	428
Tableau 348 – Builtin XMTR_IGNORE_ALL_RESPONSE_CODES	428
Tableau 349 – Builtin XMTR_IGNORE_COMM_ERROR	429
Tableau 350 – Builtin XMTR_IGNORE_COMM_STATUS	429
Tableau 351 – Builtin XMTR_IGNORE_DATA	430

Tableau 352 – Builtin XMTR_IGNORE_DEVICE_STATUS	430
Tableau 353 – Builtin XMTR_IGNORE_NO_DEVICE	431
Tableau 354 – Builtin XMTR_IGNORE_RESPONSE_CODE	431
Tableau 355 – Builtin XMTR_RETRY_ON_ALL_COMM_STATUS	432
Tableau 356 – Builtin XMTR_RETRY_ON_ALL_DATA	432
Tableau 357 – Builtin XMTR_RETRY_ON_ALL_DEVICE_STATUS	433
Tableau 358 – Builtin XMTR_RETRY_ON_ALL_RESPONSE_CODES	433
Tableau 359 – Builtin XMTR_RETRY_ON_COMM_ERROR	434
Tableau 360 – Builtin XMTR_RETRY_ON_COMM_STATUS	435
Tableau 361 – Builtin XMTR_RETRY_ON_DATA	435
Tableau 362 – Builtin XMTR_RETRY_ON_DEVICE_STATUS	436
Tableau 363 – Builtin XMTR_RETRY_ON_NO_DEVICE	436
Tableau 364 – Builtin XMTR_RETRY_ON_RESPONSE_CODE	437
Tableau 365 – Contenu du Tableau de description de codes de retour	437
Tableau 366 – Descriptions de codes de retour	438
Tableau 367 – Descriptions de codes de retour	438
Tableau A.1 – Profil de Builtin pour PROFIBUS et PROFINET	439
Tableau A.2 – Profil de Builtin pour FOUNDATION fieldbus	448
Tableau A.3 – Profil de Builtin pour HART	456
Tableau A.4 – Profil de Builtin pour les serveurs de communication	464

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

**BLOCS FONCTIONNELS (FB) POUR LES PROCÉDÉS INDUSTRIELS ET
LE LANGAGE DE DESCRIPTION ÉLECTRONIQUE DE PRODUIT (EDDL) –****Partie 5: Bibliothèque de Builtin EDDL****AVANT-PROPOS**

- 1) La Commission Electrotechnique Internationale (IEC) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de l'IEC). L'IEC a pour objet de favoriser la coopération internationale pour toutes les questions de normalisation dans les domaines de l'électricité et de l'électronique. À cet effet, l'IEC – entre autres activités – publie des Normes internationales, des Spécifications techniques, des Rapports techniques, des Spécifications accessibles au public (PAS) et des Guides (ci-après dénommés "Publication(s) de l'IEC"). Leur élaboration est confiée à des comités d'études, aux travaux desquels tout Comité national intéressé par le sujet traité peut participer. Les organisations internationales, gouvernementales et non gouvernementales, en liaison avec l'IEC, participent également aux travaux. L'IEC collabore étroitement avec l'Organisation Internationale de Normalisation (ISO), selon des conditions fixées par accord entre les deux organisations.
- 2) Les décisions ou accords officiels de l'IEC concernant les questions techniques représentent, dans la mesure du possible, un accord international sur les sujets étudiés, étant donné que les Comités nationaux de l'IEC intéressés sont représentés dans chaque comité d'études.
- 3) Les Publications de l'IEC se présentent sous la forme de recommandations internationales et sont agréées comme telles par les Comités nationaux de l'IEC. Tous les efforts raisonnables sont entrepris afin que l'IEC s'assure de l'exactitude du contenu technique de ses publications; l'IEC ne peut pas être tenue responsable de l'éventuelle mauvaise utilisation ou interprétation qui en est faite par un quelconque utilisateur final.
- 4) Dans le but d'encourager l'uniformité internationale, les Comités nationaux de l'IEC s'engagent, dans toute la mesure possible, à appliquer de façon transparente les Publications de l'IEC dans leurs publications nationales et régionales. Toutes divergences entre toutes Publications de l'IEC et toutes publications nationales ou régionales correspondantes doivent être indiquées en termes clairs dans ces dernières.
- 5) L'IEC elle-même ne fournit aucune attestation de conformité. Des organismes de certification indépendants fournissent des services d'évaluation de conformité et, dans certains secteurs, accèdent aux marques de conformité de l'IEC. L'IEC n'est responsable d'aucun des services effectués par les organismes de certification indépendants.
- 6) Tous les utilisateurs doivent s'assurer qu'ils sont en possession de la dernière édition de cette publication.
- 7) Aucune responsabilité ne doit être imputée à l'IEC, à ses administrateurs, employés, auxiliaires ou mandataires, y compris ses experts particuliers et les membres de ses comités d'études et des Comités nationaux de l'IEC, pour tout préjudice causé en cas de dommages corporels et matériels, ou de tout autre dommage de quelque nature que ce soit, directe ou indirecte, ou pour supporter les coûts (y compris les frais de justice) et les dépenses découlant de la publication ou de l'utilisation de cette Publication de l'IEC ou de toute autre Publication de l'IEC, ou au crédit qui lui est accordé.
- 8) L'attention est attirée sur les références normatives citées dans cette publication. L'utilisation de publications référencées est obligatoire pour une application correcte de la présente publication.
- 9) L'attention est attirée sur le fait que certains des éléments de la présente Publication de l'IEC peuvent faire l'objet de droits de brevet. L'IEC ne saurait être tenue pour responsable de ne pas avoir identifié de tels droits de brevets et de ne pas avoir signalé leur existence.

La Norme internationale IEC 61804-5 a été établie par le sous-comité 65E: Les dispositifs et leur intégration dans les systèmes de l'entreprise, du comité d'études 65 de l'IEC: Mesure, commande et automation dans les processus industriels.

Le texte de cette norme est issu des documents suivants:

FDIS	Rapport de vote
65E/450/FDIS	65E/461/RVD

Le rapport de vote indiqué dans le Tableau ci-dessus donne toute information sur le vote ayant abouti à l'approbation de cette norme.

Cette publication a été rédigée selon les Directives ISO/IEC, Partie 2.

Une liste de toutes les parties de la série IEC 61804, publiées sous le titre général *Blocs fonctionnels (FB) pour les procédés industriels et le langage de description électronique de produit (EDDL)*, peut être consultée sur le site web de l'IEC.

Les futures normes de cette série porteront dorénavant le nouveau titre général cité ci-dessus. Le titre des normes existant déjà dans cette série sera mis à jour lors de la prochaine édition.

Le comité a décidé que le contenu de cette publication ne sera pas modifié avant la date de stabilité indiquée sur le site web de l'IEC sous "<http://webstore.iec.ch>" dans les données relatives à la publication recherchée. A cette date, la publication sera

- reconduite,
- supprimée,
- remplacée par une édition révisée, ou
- amendée.

INTRODUCTION

La présente partie de l'IEC 61804 est une extension de l'IEC 61804-3. Dans les éditions précédentes de l'IEC 61804-3, la bibliothèque de Builtin EDDL faisait partie intégrante du document IEC 61804-3 Langage de description électronique de produit (EDDL). Pour des raisons de problèmes d'édition, la bibliothèque de Builtin EDDL est désormais intégrée à cette partie distincte de l'IEC 61804.

IECNORM.COM: Click to view the full PDF of IEC 61804-5:2015

Withd2W

BLOCS FONCTIONNELS (FB) POUR LES PROCÉDÉS INDUSTRIELS ET LE LANGAGE DE DESCRIPTION ÉLECTRONIQUE DE PRODUIT (EDDL) –

Partie 5: Bibliothèque de Builtin EDDL

1 Domaine d'application

La présente partie de l'IEC 61804 spécifie la bibliothèque de Builtin EDDL et fournit les profils des divers bus de terrain.

2 Références normatives

Les documents suivants sont cités en référence de manière normative, en intégralité ou en partie, dans le présent document et sont indispensables pour son application. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

IEC 61804-3:2015, *Blocs fonctionnels (FB) pour les procédés industriels et le langage de description électronique de produit (EDDL) – Partie 3: Sémantique et syntaxe EDDL*

IEEE 754, *Standard for Binary Floating-Point Arithmetic for microprocessor systems* (disponible en anglais uniquement)

3 Termes, définitions, acronymes et abréviations

3.1 Termes et définitions

Pour les besoins du présent document, les termes et définitions donnés dans l'IEC 61804-3 s'appliquent.

3.2 Acronymes et abréviations

CP	Communication profile (Profil de communication)
CPF	Communication profile family (Famille de profils de communication)
DD	Device description (Description de produit)
EDD	Electronic Device Description (Description électronique de produit)
EDDL	Electronic Device Description Language (Langage de description électronique de produit)
HCF	HART® ¹ Communication Foundation
E/S (I/O)	Entrée/sortie (Input/output)
ID	Identifiant
PI	PROFIBUS & PROFINET ² International

¹ HART® est une marque déposée du consortium HART Communication Foundation (HCF) (organisation à but non lucratif). Cette information est donnée à l'intention des utilisateurs de la présente norme et ne signifie nullement que l'IEC approuve ou recommande le détenteur de l'appellation commerciale ou l'un quelconque de ses produits. La conformité à ce profil ne nécessite pas l'utilisation de l'appellation commerciale. L'utilisation de l'appellation commerciale nécessite l'autorisation du propriétaire de la marque.

PNO PROFIBUS Nutzerorganisation

4 Bibliothèque de Builtin EDDL

4.1 Généralités

Le présent Article 4 décrit les sous-programmes Builtin, qui peuvent être utilisés dans une EDD. Les Builtin prennent en charge les interactions avec l'utilisateur ou le dispositif, voir Annexe A. Des interactions types avec l'utilisateur sont, par exemple, des entrées par l'utilisateur de valeurs de VARIABLE ou d'accusés de réception. Des interactions types avec le dispositif sont, par exemple, l'envoi des commandes au dispositif ou des codes de réponse d'interprétation et des octets de statut. L'Article 4 contient la bibliothèque des Builtin.

Outre les Builtin spécifiés dans ce document, des Builtin spécifiques à l'utilisateur peuvent être utilisés. Ces Builtin doivent être intégrés dans l'application EDD.

NOTE Les Builtin sont répertoriés dans l'ordre alphabétique.

4.2 Conventions pour les descriptions de Builtin

Les conventions suivantes s'appliquent pour les définitions de Builtin.

- Le Tableau 1 définit le format pour les tableaux d'élément lexical de Builtin.
- Le Tableau 2 définit le contenu des colonnes du Tableau 1.
- Les constructions et les attributs de base EDDL référencés dans le présent document sont présentés en majuscules.

Tableau 1 – Format pour les tableaux d'élément lexical de Builtin

Nom du paramètre	Type	Direction	Usage	Description

² PROFIBUS & PROFINET sont des appellations commerciales de PROFIBUS Nutzerorganisation e.V. (PNO) filiale de PROFIBUS & PROFINET International. PNO est une organisation à but non lucratif pour le support des technologies de bus de terrain PROFIBUS & PROFINET. Cette information est donnée à l'intention des utilisateurs de la présente norme et ne signifie nullement que l'IEC approuve ou recommande le détenteur de l'appellation commerciale ou l'un quelconque de ses produits. La conformité à ce profil ne nécessite pas l'utilisation de l'appellation commerciale. L'utilisation de l'appellation commerciale nécessite l'autorisation du propriétaire de la marque.

Tableau 2 – Contenu du Tableau d'élément lexical

Colonne	Texte	Signification
Nom du paramètre	<name>	Nom du paramètre formel.
	"<return>"	Valeur de retour du Builtin.
Type	data type	Type de données abstrait (basique ou dérivé) du paramètre formel.
Direction	I	Paramètre d'entrée.
	O	Un paramètre de sortie est passé de manière implicite par la référence de sorte que la fonction de Builtin peut modifier la valeur. La valeur de retour d'un Builtin est passée par valeur, le cas échéant.
	I/O	Un paramètre d'entrée-sortie est passé de manière implicite par la référence de sorte que la fonction de Builtin peut modifier la valeur.
	—	Aucune valeur n'est retournée (VOID).
Usage	m	Le paramètre formel est obligatoire.
	o	Le paramètre formel est facultatif.
	c	Le paramètre formel est conditionnel.
	—	Aucune valeur n'est retournée (VOID).
Description	<text>	Texte descriptif pour le paramètre.
	—	Aucune valeur n'est retournée (VOID).

4.3 Catégories de Builtin

4.3.1 Vue d'ensemble

Les catégories de Builtin spéciales sont les suivantes:

- Builtin de l'interface utilisateur
- Builtin de communication
- Builtin d'action

Des restrictions s'appliquent à l'usage des Builtin, selon leur catégorie (voir le Tableau 3).

Tableau 3 – Usage des Builtin

Usage des Builtin	Builtin de l'interface utilisateur	Builtin de communication	Builtin d'action
Méthodes référencées dans les menus de l'interface utilisateur	Autorisé	Autorisé	Non autorisé
PRE_EDIT_ACTIONS	Autorisé	Non recommandé	Autorisé ^a
POST_EDIT_ACTIONS	Autorisé	Non recommandé	Autorisé ^a
REFRESH_ACTIONS	Non recommandé	GRAPH et CHART: autorisés Autres: non recommandés	Autorisé ^a
PRE_READ_ACTIONS	Non recommandé	Autorisé	Autorisé ^a
POST_READ_ACTIONS	Non recommandé	Autorisé	Autorisé ^a
PRE_WRITE_ACTIONS	Non recommandé	Autorisé	Autorisé ^a
POST_WRITE_ACTIONS	Non recommandé	Autorisé	Autorisé ^a
INIT_ACTIONS	Non recommandé	Autorisé	Non autorisé
EXIT_ACTIONS	Non recommandé	Autorisé	Non autorisé
^a Doit être utilisé uniquement pour les actions appelées par une VARIABLE.			

4.3.2 Builtin de l'interface utilisateur

Les builtin de l'interface utilisateur affichent l'information et, de plus, certains d'entre eux exigent une interaction de l'utilisateur (voir le Tableau 4).

Tableau 4 – Builtin de l'interface utilisateur

Nom du Builtin
ACKNOWLEDGE
acknowledge
DELAY
delay
DISPLAY
display
display_bitenum
display_builtin_error
display_comm_error
display_comm_status
display_device_status
display_dynamics
display_dynamics2
display_message
display_message2
display_response_code
display_response_status
display_xmtr_status
edit_device_value
edit_device_value2
edit_local_value
edit_local_value2
get_acknowledgement2
get_acknowledgement
GET_DEV_VAR_VALUE
get_dev_var_value
GET_LOCAL_VAR_VALUE
get_local_var_value
MenuDisplay
PUT_MESSAGE
put_message
SELECT_FROM_LIST
select_from_list
select_from_menu
select_from_menu2

4.3.3 Builtin de communication

Les Builtin de communication permettent aux méthodes de communiquer avec le dispositif ou de contrôler la communication avec ce dernier (voir le Tableau 5).

Tableau 5 – Builtin de communication

Nom du Builtin
abort_on_all_comm_errors
ABORT_ON_ALL_COMM_STATUS
ABORT_ON_ALL_DEVICE_STATUS
ABORT_ON_ALL_RESPONSE_CODES
abort_on_all_response_codes
ABORT_ON_COMM_ERROR
abort_on_comm_error
ABORT_ON_COMM_STATUS
ABORT_ON_DEVICE_STATUS
ABORT_ON_NO_DEVICE
ABORT_ON_RESPONSE_CODE
abort_on_response_code
closeTransferPort
ext_send_command
ext_send_command_trans
get_transfer_status
IGNORE_ALL_COMM_STATUS
IGNORE_ALL_DEVICE_STATUS
IGNORE_ALL_RESPONSE_CODES
IGNORE_COMM_ERROR
IGNORE_COMM_STATUS
IGNORE_DEVICE_STATUS
IGNORE_NO_DEVICE
IGNORE_RESPONSE_CODE
openTransferPort
read_value
read_value2
ReadCommand
readItemFromDevice
retry_on_all_comm_errors
RETRY_ON_ALL_COMM_STATUS
RETRY_ON_ALL_DEVICE_STATUS
RETRY_ON_ALL_RESPONSE_CODES
retry_on_all_response_codes
RETRY_ON_COMM_ERROR
retry_on_comm_error
RETRY_ON_COMM_STATUS
RETRY_ON_DEVICE_STATUS
RETRY_ON_NO_DEVICE

Nom du Builtin
RETRY_ON_RESPONSE_CODE
retry_on_response_code
send
send_all_values
send_command
send_command_trans
send_on_exit
send_trans
send_value
send_value2
WriteCommand
writelnItemToDevice
XMTR_ABORT_ON_ALL_COMM_STATUS
XMTR_ABORT_ON_ALL_DATA
XMTR_ABORT_ON_ALL_DEVICE_STATUS
XMTR_ABORT_ON_ALL_RESPONSE_CODES
XMTR_ABORT_ON_COMM_ERROR
XMTR_ABORT_ON_COMM_STATUS
XMTR_ABORT_ON_DATA
XMTR_ABORT_ON_DEVICE_STATUS
XMTR_ABORT_ON_NO_DEVICE
XMTR_ABORT_ON_RESPONSE_CODE
XMTR_IGNORE_ALL_COMM_STATUS
XMTR_IGNORE_ALL_DATA
XMTR_IGNORE_ALL_DEVICE_STATUS
XMTR_IGNORE_ALL_RESPONSE_CODES
XMTR_IGNORE_COMM_ERROR
XMTR_IGNORE_COMM_STATUS
XMTR_IGNORE_DATA
XMTR_IGNORE_DEVICE_STATUS
XMTR_IGNORE_NO_DEVICE
XMTR_IGNORE_RESPONSE_CODE
XMTR_RETRY_ON_ALL_COMM_STATUS
XMTR_RETRY_ON_ALL_DATA
XMTR_RETRY_ON_ALL_DEVICE_STATUS
XMTR_RETRY_ON_ALL_RESPONSE_CODE
XMTR_RETRY_ON_ALL_RESPONSE_CODES
XMTR_RETRY_ON_COMM_ERROR
XMTR_RETRY_ON_COMM_STATUS
XMTR_RETRY_ON_DATA
XMTR_RETRY_ON_DEVICE_STATUS
XMTR_RETRY_ON_NO_DEVICE
XMTR_RETRY_ON_RESPONSE_CODE

4.3.4 Builtin d'action

Les Builtin d'action permettent d'accéder à la valeur de la variable qui a appelé l'action (voir le Tableau 6). Ces Builtin peuvent ne pas être utilisés dans d'autres méthodes ou dans des actions appelées par un MENU.

Tableau 6 – Builtin d'action

Nom du Builtin
fgetval
fsetval
get_date
get_double
get_float
get_signed
get_string
get_unsigned
igetval
isetval
put_date
put_double
put_float
put_signed
put_string
put_unsigned
sgetval
ssetval

4.4 Builtin _ERROR

Objet

Le Builtin _ERROR place une information textuelle dans un journal. Cette information est placée dans la catégorie d'erreur. Le Builtin _TRACE et le Builtin _WARNING écrivent des informations dans ce même journal.

Toute référence de variable contenue au format chaîne est développée en valeur actuelle. Des variables locales de méthode et des VARIABLES peuvent être référencées dans le paramètre de message. Pour plus d'informations, consulter le Builtin PUT_MESSAGE.

Structure lexicale

_ERROR(text)

Les éléments lexicaux du Builtin _ERROR sont présentés dans le Tableau 7.

Tableau 7 – Builtin _ERROR

Nom du paramètre	Type	Direction	Usage	Description
<text>	DD_STRING	I	m	Information textuelle à placer dans le journal.
<return>	VOID	—	—	—

4.5 Builtin _TRACE

Objet

Le Builtin _TRACE place une information textuelle dans un journal. Cette information est placée dans la catégorie trace. Le Builtin _ERROR et le Builtin _WARNING écrivent des informations dans ce même journal.

Toute référence de variable contenue au format chaîne est développée en valeur actuelle. Des variables locales de méthode et des VARIABLES peuvent être référencées dans le paramètre de message. Pour plus d'informations, consulter le Builtin PUT_MESSAGE.

Structure lexicale

_TRACE(text)

Les éléments lexicaux du Builtin _TRACE sont présentés dans le Tableau 8.

Tableau 8 – Builtin _TRACE

Nom du paramètre	Type	Direction	Usage	Description
text	DD_STRING	I	m	Information textuelle à placer dans le journal.
<return>	VOID	—	—	—

4.6 Builtin _WARNING

Objet

Le Builtin _WARNING (avertissement) place une information textuelle dans un journal. Cette information est placée dans la catégorie warning (avertissement). Le Builtin _ERROR et le Builtin _TRACE écrivent des informations dans ce même journal.

Toute référence de variable contenue au format chaîne est développée en valeur actuelle. Des variables locales de méthode et des VARIABLES peuvent être référencées dans le paramètre de message. Pour plus d'informations, consulter le Builtin PUT_MESSAGE.

Structure lexicale

_WARNING(text)

Les éléments lexicaux du Builtin _WARNING sont présentés dans le Tableau 9.

Tableau 9 – Builtin _WARNING

Nom du paramètre	Type	Direction	Usage	Description
text	DD_STRING	I	m	Information textuelle à placer dans le journal.
<return>	VOID	—	—	—

4.7 Builtin abort

Objet

Le Builtin abort (interrompre) affiche un message indiquant que la méthode a été interrompue et attend un accusé de réception de l'utilisateur. Une fois l'accusé de réception confirmé, le système exécute des méthodes d'interruption dans la liste de méthodes d'interruption et quitte la méthode.

Structure lexicale

abort (VOID)

L'élément lexical du Builtin abort est présenté dans le Tableau 10.

Tableau 10 – Builtin abort

Nom du paramètre	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.8 Builtin abort_on_all_comm_errors

Objet

Le Builtin abort_on_all_comm_errors spécifie la méthode à interrompre en cas de réception d'erreur de communication.

Lorsqu'une méthode est démarrée, elle n'est pas interrompue sur une erreur de communication à moins qu'abort_on_all_comm_errors soit appelé. Cela évite à une méthode de devoir gérer des codes d'erreur de communication en interrompant la méthode si elle reçoit une erreur de communication.

Après cet appel, tout appel de Builtin qui effectue une demande de communication qui retourne une erreur conduit à l'interruption de la méthode.

Structure lexicale

abort_on_all_comm_errors (VOID)

L'élément lexical du Builtin abort_on_all_comm_errors est présenté dans le Tableau 11.

Tableau 11 – Builtin abort_on_all_comm_errors

Nom du paramètre	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.9 Builtin ABORT_ON_ALL_COMM_STATUS

Objet

Le Builtin ABORT_ON_ALL_COMM_STATUS définit tous les bits dans le masque communication status abort (interruption statut de communication). Cela conduit le système à interrompre la méthode actuelle si le dispositif retourne une valeur de statut de communication. Le statut de communication est défini comme étant le premier octet de données retourné dans une transaction, lorsque le bit 7 de cet octet est défini.

Les masques de relance et d'interruption sont réinitialisés à leurs valeurs par défaut au début de chaque méthode, de sorte que la nouvelle valeur du masque est valide uniquement pendant la méthode actuelle. Voir le Builtin ABORT_ON_RESPONSE_CODE pour une liste des masques disponibles, et leurs valeurs par défaut.

Le masque affecté par cette fonction est utilisé par les Builtin send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, et display.

Structure lexicale

ABORT_ON_ALL_COMM_STATUS (VOID)

L'élément lexical du Builtin ABORT_ON_ALL_COMM_STATUS est présenté dans le Tableau 12.

Tableau 12 – Builtin ABORT_ON_ALL_COMM_STATUS

Nom du paramètre	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.10 Builtin ABORT_ON_ALL_DEVICE_STATUS

Objet

Le Builtin ABORT_ON_ALL_DEVICE_STATUS définit tous les bits dans le masque device status abort (interruption statut du dispositif). Cela conduit le système à interrompre la méthode actuelle si le dispositif retourne une valeur de statut du dispositif. Le statut du dispositif est défini comme étant le deuxième octet de données retourné dans une transaction.

Les masques de relance et d'interruption sont réinitialisés à leurs valeurs par défaut au début de chaque méthode, de sorte que la nouvelle valeur du masque est valide uniquement pendant la méthode actuelle. Voir la fonction send_command pour l'implémentation des masques. Voir ABORT_ON_RESPONSE_CODE pour une liste des masques disponibles, et leurs valeurs par défaut.

Le masque affecté par cette fonction est utilisé par les Builtin send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, et display.

Structure lexicale

ABORT_ON_ALL_DEVICE_STATUS (VOID)

L'élément lexical du Builtin ABORT_ON_ALL_DEVICE_STATUS est présenté dans le Tableau 13.

Tableau 13 – Builtin ABORT_ON_ALL_DEVICE_STATUS

Nom du paramètre	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.11 Builtin ABORT_ON_ALL_RESPONSE_CODES

Objet

Le Builtin ABORT_ON_ALL_RESPONSE_CODES définit tous les bits dans le masque response code abort (interruption code de réponse). Cela conduit le système à interrompre la méthode actuelle si le dispositif retourne une valeur code de réponse.

Le code de réponse est défini comme étant le premier octet de données retourné dans une transaction, lorsque le bit 7 de cet octet est 0.

Les masques de relance et d'interruption sont réinitialisés à leurs valeurs par défaut au début de chaque méthode, de sorte que la nouvelle valeur du masque est valide uniquement pendant la méthode actuelle. Voir la fonction de commande d'envoi pour l'implémentation des masques. Voir ABORT_ON_RESPONSE_CODE pour une liste des masques disponibles, et leurs valeurs par défaut.

Le masque affecté par cette fonction est utilisé par les Builtin send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, et display.

Structure lexicale

ABORT_ON_ALL_RESPONSE_CODES (VOID)

L'élément lexical du Builtin ABORT_ON_ALL_RESPONSE_CODES est présenté dans le Tableau 14.

Tableau 14 – Builtin ABORT_ON_ALL_RESPONSE_CODES

Nom du paramètre	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.12 Builtin abort_on_all_response_codes

Objet

Le Builtin abort_on_all_response_codes conduit à l'interruption de la méthode si celle-ci reçoit un code réponse différent de zéro, après l'appel d'un Builtin qui conduit à une demande à un dispositif. Ce Builtin évite à une méthode de devoir gérer des codes de réponse d'erreur. Lorsqu'une méthode est démarrée, elle n'est pas interrompue sur un code réponse jusqu'à ce que ce Builtin soit appelé. Un code réponse est une valeur représentant une condition d'erreur spécifique à une application. Une valeur code de réponse égale à zéro n'est pas considérée comme un code réponse d'erreur. Chaque code de réponse et type de code de réponse est défini dans l'EDDL.

Structure lexicale

abort_on_all_response_codes (VOID)

L'élément lexical du Builtin abort_on_all_response_codes est présenté dans le Tableau 15.

Tableau 15 – Builtin abort_on_all_response_codes

Nom du paramètre	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.13 Builtin ABORT_ON_COMM_ERROR

Objet

Le Builtin ABORT_ON_COMM_ERROR définit le masque no comm error (pas d'erreur de communication) de sorte que la méthode soit interrompue si une erreur de communication est détectée lors de l'envoi d'une commande.

Les masques de relance et d'interruption sont réinitialisés à leurs valeurs par défaut au début de chaque méthode, de sorte que la nouvelle valeur du masque est valide uniquement pendant la méthode actuelle. Voir la fonction send_command pour l'implémentation des masques. Voir le Builtin ABORT_ON_RESPONSE_CODE pour une liste des masques disponibles et leurs valeurs par défaut.

Le masque affecté par cette fonction est utilisé par les Builtin send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, et display.

Structure lexicale

ABORT_ON_COMM_ERROR(VOID)

L'élément lexical du Builtin ABORT_ON_COMM_ERROR est présenté dans le Tableau 16.

Tableau 16 – Builtin ABORT_ON_COMM_ERROR

Nom du paramètre	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.14 Builtin abort_on_comm_error

Objet

Le Builtin abort_on_comm_error conduit à l'interruption de la méthode si celle-ci reçoit une erreur de communication spécifique.

Lorsqu'une méthode est démarrée, celle-ci n'est pas interrompue sur une erreur de communication spécifique à moins que le Builtin abort_on_comm_error ne soit appelé.

NOTE Une erreur de communication de zéro n'est pas considérée comme une erreur de communication.

Ce Builtin permet à l'auteur de la méthode de sélectionner les erreurs de communication qu'il convient que la méthode gère et cela évite à une méthode de devoir gérer un code d'erreur de communication spécifique.

Les codes d'erreur de communication sont définis par les consortiums. Leurs chaînes associées peuvent être définies dans un dictionnaire de texte.

Structure lexicale

abort_on_comm_error(error)

Les éléments lexicaux du Builtin abort_on_comm_error sont présentés dans le Tableau 17.

Tableau 17 – Builtin abort_on_comm_error

Nom du paramètre	Type	Direction	Usage	Description
error	unsigned long	I	m	Code d'erreur de communication.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.15 Builtin ABORT_ON_COMM_STATUS

Objet

Le Builtin ABORT_ON_COMM_STATUS définit le(s) bit(s) correct(s) dans le masque d'interruption comm status (statut de communication) de sorte que la comm_status_value spécifiée conduise à l'interruption de la méthode. Le statut de communication est défini comme étant le premier octet de données retourné dans une transaction, lorsque le bit 7 de cet octet est positionné.

Les masques de relance et d'interruption sont réinitialisés à leurs valeurs par défaut au début de chaque méthode, de sorte que la nouvelle valeur du masque est valide uniquement pendant la méthode actuelle. Voir la fonction de commande d'envoi pour l'implémentation des masques. Voir ABORT_ON_RESPONSE_CODE pour une liste des masques disponibles, et leurs valeurs par défaut.

Le masque affecté par cette fonction est utilisé par les Builtin send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, et display.

Structure lexicale

ABORT_ON_COMM_STATUS(comm_status)

Les éléments lexicaux du Builtin ABORT_ON_COMM_STATUS sont présentés dans le Tableau 18.

Tableau 18 – Builtin ABORT_ON_COMM_STATUS

Nom du paramètre	Type	Direction	Usage	Description
comm_status	unsigned char	I	m	Masque binaire pour le statut de communication.
<return>	VOID	—	—	—

4.16 Builtin ABORT_ON_DEVICE_STATUS

Objet

Le Builtin ABORT_ON_DEVICE_STATUS définit le(s) bit(s) correct(s) dans le masque device status abort (interruption statut du dispositif) de sorte que la valeur de statut du dispositif spécifiée conduise à l'interruption de la méthode. Le statut du dispositif est défini comme étant le deuxième octet de données retourné dans une transaction.

Les masques de relance et d'interruption sont réinitialisés à leurs valeurs par défaut au début de chaque méthode, de sorte que la nouvelle valeur du masque est valide uniquement pendant la méthode actuelle. Voir la fonction de commande d'envoi pour l'implémentation des masques. Voir ABORT_ON_RESPONSE_CODE pour une liste des masques disponibles, et leurs valeurs par défaut.

Le masque affecté par cette fonction est utilisé par les Builtin send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, et display.

Structure lexicale

ABORT_ON_DEVICE_STATUS(device_status)

Les éléments lexicaux du Builtin ABORT_ON_DEVICE_STATUS sont présentés dans le Tableau 19.

Tableau 19 – Builtin ABORT_ON_DEVICE_STATUS

Nom du paramètre	Type	Direction	Usage	Description
device_status	unsigned char	I	m	Masque binaire pour le statut du dispositif.
<return>	VOID	—	—	—

4.17 Builtin ABORT_ON_NO_DEVICE

Objet

Le Builtin ABORT_ON_NO_DEVICE définit le masque no devices (pas de dispositifs) de sorte que la méthode soit interrompue si aucun dispositif n'est détecté lors de l'envoi d'une commande.

Les masques de relance et d'interruption sont réinitialisés à leurs valeurs par défaut au début de chaque méthode, de sorte que la nouvelle valeur du masque est valide uniquement pendant la méthode actuelle. Voir la fonction de commande d'envoi pour l'implémentation des masques. Voir ABORT_ON_RESPONSE_CODE pour une liste des masques disponibles, et leurs valeurs par défaut.

Le masque affecté par cette fonction est utilisé par les Builtin send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, et display.

Structure lexicale

ABORT_ON_NO_DEVICE (VOID)

L'élément lexical du Builtin ABORT_ON_NO_DEVICE est présenté dans le Tableau 20.

Tableau 20 – Builtin ABORT_ON_NO_DEVICE

Nom du paramètre	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.18 Builtin ABORT_ON_RESPONSE_CODE

Objet

Le Builtin ABORT_ON_RESPONSE_CODE définit le(s) bit(s) correct(s) dans le masque response code abort (interruption de code de réponse) de sorte que la valeur de code de réponse spécifiée conduise à l'interruption de la méthode. Le code de réponse est défini comme étant le premier octet de données retourné dans une transaction, lorsque le bit 7 de cet octet est 0.

Les masques de relance et d'interruption sont réinitialisés à leurs valeurs par défaut au début de chaque méthode, de sorte que la nouvelle valeur du masque est valide uniquement pendant la méthode actuelle.

Une liste de l'ensemble des masques disponibles est présentée ci-dessous. Les valeurs par défaut des masques sont les suivantes:

- response_code_abort_mask = ACCESS_RESTRICTED | CMD_NOT_IMPLEMENTED
- response_code_retry_mask = NO BITS SET
- cmd_48_response_code_abort_mask = ACCESS_RESTRICTED | CMD_NOT_IMPLEMENTED
- cmd_48_response_code_retry_mask = NO BITS SET
- device_status_abort_mask = DEVICE_MALFUNCTION
- device_status_retry_mask = NO BITS SET
- cmd_48_device_status_abort_mask = DEVICE_MALFUNCTION
- cmd_48_device_status_retry_mask = NO BITS SET
- comm_status_abort_mask = NO BITS SET
- comm_status_retry_mask = ALL BITS SET
- cmd_48_comm_status_abort_mask = NO BITS SET
- cmd_48_comm_status_retry_mask = ALL BITS SET
- no_device_abort_mask = ALL BITS SET
- comm_error_abort_mask = ALL BITS SET

- comm_error_retry_mask = NO BITS SET
- cmd_48_no_device_abort_mask = ALL BITS SET
- cmd_48_comm_error_abort_mask = ALL BITS SET
- cmd_48_comm_error_retry_mask = NO BITS SET
- cmd_48_data_abort_mask = NO BITS SET
- cmd_48_data_retry_mask = NO BITS SET

Le masque affecté par cette fonction est utilisé par les Builtin send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status, et display.

Structure lexicale

ABORT_ON_RESPONSE_CODE(response_code)

Les éléments lexicaux du Builtin ABORT_ON_RESPONSE_CODE sont présentés dans le Tableau 21.

Tableau 21 – Builtin ABORT_ON_RESPONSE_CODE

Nom du paramètre	Type	Direction	Usage	Description
response_code	int	I	m	Masque binaire pour le code de réponse.
<return>	VOID	—	—	—

4.19 Builtin abort_on_response_code

Objet

Le Builtin abort_on_response_code conduit à l'interruption de la méthode après réception du code de réponse spécifié.

Le Builtin conduit à l'interruption de la méthode si celle-ci reçoit un code de réponse spécifique. Ce Builtin évite à une méthode de devoir gérer un code de réponse spécifique.

Lorsqu'une méthode est démarrée, celle-ci n'est pas interrompue sur un code de réponse spécifique à moins que ce Builtin ne soit appelé. Un code de réponse est une valeur représentant une condition d'erreur spécifique à une application.

NOTE Une valeur de code de réponse zéro n'est pas considérée comme un code de réponse d'erreur. Chaque type de code de réponse est défini dans l'EDD.

Structure lexicale

abort_on_response_code(code)

Les éléments lexicaux du Builtin abort_on_response_code sont présentés dans le Tableau 22.

Tableau 22 – Builtin abort_on_response_code

Nom du paramètre	Type	Direction	Usage	Description
code	unsigned long	I	m	Masque binaire pour le code de réponse.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.20 Builtin abortTransferPort

Objet

Le Builtin abortTransferPort interrompt le Block Data Transfer (transfert de données de blocs) indiqué par le descripteur.

Structure lexicale

abortTransferPort(handle)

Les éléments lexicaux du Builtin abortTransferPort sont présentés dans le Tableau 23.

Tableau 23 – Builtin abortTransferPort

Nom du paramètre	Type	Direction	Usage	Description
handle	unsigned long	I	m	Descripteur du transfert de blocs.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau 367

4.21 Builtin abs

Objet

Le Builtin abs retourne la valeur absolue d'une valeur à virgule flottante.

Structure lexicale

abs(x)

Les éléments lexicaux du Builtin abs sont présentés dans le Tableau 24.

Tableau 24 – Builtin abs

Nom du paramètre	Type	Direction	Usage	Description
x	double	I	m	Valeur à virgule flottante.
<return>	double	O	m	Valeur absolue.

4.22 Builtin ACKNOWLEDGE

Objet

Le Builtin ACKNOWLEDGE (accuser réception) affiche l'invite et attend la pression de la touche Entrée. Contrairement au Builtin acknowledge, l'invite peut NE PAS contenir de VARIABLES du dispositif incorporées.

Structure lexicale

ACKNOWLEDGE(prompt)

Les éléments lexicaux du Builtin ACKNOWLEDGE sont présentés dans le Tableau 25.

Tableau 25 – Builtin ACKNOWLEDGE

Nom du paramètre	Type	Direction	Usage	Description
prompt	char[]	I	m	Message à afficher.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau 367

4.23 Builtin acknowledge

Objet

Le Builtin acknowledge affiche l'invite et attend l'accusé de réception de l'utilisateur. L'invite peut contenir des valeurs de variables locales et/ou du dispositif.

Structure lexicale

`acknowledge(prompt, global_var_ids)`

Les éléments lexicaux du Builtin acknowledge sont présentés dans le Tableau 26.

Tableau 26 – Builtin acknowledge

Nom du paramètre	Type	Direction	Usage	Description
prompt	char[]	I	m	Message à afficher.
global_var_ids	array of int	I	m	Matrice contenant des identifiants uniques d'instances de VARIABLE.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau 367.

4.24 Builtin acos

Objet

Le Builtin acos retourne l'arc cosinus d'une valeur à virgule flottante.

Structure lexicale

`acos(x)`

Les éléments lexicaux du Builtin acos sont présentés dans le Tableau 27.

Tableau 27 – Builtin acos

Nom du paramètre	Type	Direction	Usage	Description
x	double	I	m	Valeur à virgule flottante.
<return>	double	O	m	Arc cosinus.

4.25 Builtin add_abort_method (version A)

Objet

Le Builtin add_abort_method ajoute une méthode à la liste de méthodes d'interruption, qui est la liste des méthodes à exécuter si la méthode actuelle est interrompue. La liste de méthodes d'interruption peut contenir jusqu'à 20 méthodes. Les méthodes sont exécutées dans l'ordre dans lequel elles sont ajoutées à la liste, et la même méthode peut être ajoutée à la liste plus d'une fois. La liste est effacée après chaque exécution de la méthode non_abort.

add_abort_method est exécuté uniquement lorsque la méthode est interrompue, et non lorsque la méthode se termine dans des conditions normales de fonctionnement. Des méthodes peuvent être interrompues en raison d'une condition de masque d'interruption lors de l'envoi d'une commande, ou lorsque le Builtin abort est appelé ou lorsque l'utilisateur interrompt la méthode.

Structure lexicale

`add_abort_method(abort_method_id)`

Les éléments lexicaux du Builtin `add_abort_method` sont présentés dans le Tableau 28.

Tableau 28 – Builtin `add_abort_method`

Nom du paramètre	Type	Direction	Usage	Description
<code>abort_method</code>	reference	I	m	Référence à une METHOD.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau 367.

4.26 Builtin `add_abort_method` (version B)

Objet

Le Builtin `add_abort_method` ajoute une méthode d'interruption à la liste de méthodes d'interruption de la méthode appelante. La liste de méthodes d'interruption est la liste des méthodes qui sont exécutées si la méthode est interrompue. La liste de méthodes d'interruption s'applique uniquement à l'exécution en cours de la méthode.

Les méthodes sont interrompues lorsque des codes de réponse ou des erreurs de communication surviennent qui ont été configurés pour déclencher une interruption via d'autres appels de Builtin, par exemple, `abort_on_all_comm_errors` et `abort_on_all_response_codes`. Des méthodes peuvent également être explicitement interrompues par un appel direct du Builtin `method_abort`, ou lorsqu'une erreur se produit. La terminaison normale d'une méthode n'est pas considérée comme une interruption.

La liste de méthodes d'interruption est initialement vide lorsqu'une méthode commence son exécution. Les méthodes d'interruption sont ajoutées à la fin de la liste et sont supprimées et exécutées depuis la tête de la liste dans l'ordre dans lequel elles apparaissent, c'est-à-dire, selon la règle premier entré/premier sorti. Une méthode d'interruption peut apparaître plus d'une fois dans la liste.

Les méthodes d'interruption peuvent être partagées entre des méthodes, ce qui signifie que deux méthodes peuvent avoir la même méthode d'interruption dans leurs listes de méthodes d'interruption respectives.

Le paramètre `method_id` est spécifié par l'auteur de la méthode en utilisant le Builtin `ITEM_ID` et le nom de la méthode. Les Builtin de méthodes d'interruption ne peuvent pas appeler `add_abort_method`.

Structure lexicale

```
add_abort_method(method_id)
```

Les éléments lexicaux du Builtin `add_abort_method` sont présentés dans le Tableau 29.

Tableau 29 – Builtin `add_abort_method`

Nom du paramètre	Type	Direction	Usage	Description
<code>method_id</code>	unsigned long	I	m	ID de la méthode.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.27 Builtin `AddTime`

Objet

Le Builtin `AddTime` ajoute un nombre de secondes à une valeur `time_t`.

Structure lexicale

AddTime(t0, seconds)

Les éléments lexicaux du Builtin AddTime sont présentés dans le Tableau 30.

Tableau 30 – Builtin AddTime

Nom du paramètre	Type	Direction	Usage	Description
t0	time_t	I	m	Valeur time_t initiale.
seconds	double	I	m	Nombre de secondes à ajouter.
<return>	time_t	O	m	Valeur time_t spécifiée.

4.28 Builtin asin

Objet

Le Builtin asin retourne l'arc sinus d'une valeur à virgule flottante.

Structure lexicale

asin(x)

Les éléments lexicaux du Builtin asin sont présentés dans le Tableau 31.

Tableau 31 – Builtin asin

Nom du paramètre	Type	Direction	Usage	Description
x	double	I	m	Valeur à virgule flottante.
<return>	double	O	m	Arc sinus.

4.29 Builtin assign

Objet

Le Builtin assign définit la valeur d'une VARIABLE à la valeur d'une autre VARIABLE. Il définit la valeur de la VARIABLE de destination à une valeur égale à la valeur de la VARIABLE source.

NOTE Le Builtin est un moyen raccourci d'effectuer un ABN_GET_TYPE_VALUE suivi par une commande put_type_value.

Les paramètres dst_id, src_id, dst_member_id, et src_member_id sont spécifiés en utilisant les Builtin ITEM_ID et MEMBER ID et la référence de la VARIABLE. La VARIABLE source et la VARIABLE de destination doivent être du même type.

Structure lexicale

assign(dst_id, dst_member_id, src_id, src_member_id)

Les éléments lexicaux du Builtin assign sont présentés dans le Tableau 32.

Tableau 32 – Builtin assign

Nom du paramètre	Type	Direction	Usage	Description
dst_id	unsigned long	I	m	ID d'item de la variable de destination.
dst_member_id	unsigned long	I	m	ID de membre de la variable de destination.
src_id	unsigned long	I	m	ID d'item de la variable source.
src_member_id	unsigned long	I	m	ID de membre de la variable source.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.30 Builtin assign_double**Objet**

Le Builtin assign_double assigne une valeur à une VARIABLE.

Structure lexicale

```
assign_double(device_var, new_value)
```

Les éléments lexicaux du Builtin assign_double sont présentés dans le Tableau 33.

Tableau 33 – Builtin assign_double

Nom du paramètre	Type	Direction	Usage	Description
device_var	reference	I	m	Nom de la variable.
new_value	double	I	m	Nouvelle valeur de la variable référencée.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau 367.

4.31 Builtin assign_float**Objet**

Le Builtin assign_float assigne une valeur à une VARIABLE.

Structure lexicale

```
assign_float(device_var, new_value)
```

Les éléments lexicaux du Builtin assign_float sont présentés dans le Tableau 34.

Tableau 34 – Builtin assign_float

Nom du paramètre	Type	Direction	Usage	Description
device_var	reference	I	m	Nom de la variable.
new_value	double	I	m	Nouvelle valeur de la variable référencée.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau 367.

4.32 Builtin assign_int**Objet**

Le Builtin assign_int assigne une valeur à une VARIABLE.

Structure lexicale

`assign_int(device_var, new_value)`

Les éléments lexicaux du Builtin `assign_int` sont présentés dans le Tableau 35.

Tableau 35 – Builtin `assign_int`

Nom du paramètre	Type	Direction	Usage	Description
<code>device_var</code>	reference	I	m	Nom de la variable.
<code>new_value</code>	int	I	m	Nouvelle valeur de la variable référencée.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau 367.

4.33 Builtin `assign_var`

Objet

Le Builtin `assign_var` assigne une valeur de la VARIABLE source à la VARIABLE de destination. Les deux variables doivent être du même type.

Structure lexicale

`assign_var(destination_var, source_var)`

Les éléments lexicaux du Builtin `assign_var` sont présentés dans le Tableau 36.

Tableau 36 – Builtin `assign_var`

Nom du paramètre	Type	Direction	Usage	Description
<code>destination_var</code>	reference	I	m	Nom de la variable de destination.
<code>source_var</code>	reference	I	m	Nom de la variable source.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau 367.

4.34 Builtin `assign2`

Objet

Le Builtin `assign2` définit la valeur d'une VARIABLE à la valeur d'une autre VARIABLE. Il définit la valeur de la VARIABLE de destination à une valeur égale à la valeur de la VARIABLE source.

Les paramètres `dst_id`, `src_id`, `dst_member_id`, et `src_member_id` sont spécifiés en utilisant les Builtin `ITEM_ID` et `MEMBER ID` et la référence de la VARIABLE. La VARIABLE source et la VARIABLE de destination doivent être du même type.

Structure lexicale

`assign2(dst_blk_id, dst_blk_num, dst_id, dst_member_id, src_blk_id, src_blk_num, src_id, src_member_id)`

Les éléments lexicaux du Builtin `assign` sont présentés dans le Tableau 37.

Tableau 37 – Builtin assign2

Nom du paramètre	Type	Direction	Usage	Description
dst_blk_id	unsigned long	I	m	ID d'item du bloc de destination.
dst_blk_num	unsigned long	I	m	Numéro d'occurrence du bloc de destination.
dst_id	unsigned long	I	m	ID d'item de la variable de destination.
dst_member_id	unsigned long	I	m	ID de membre de la variable de destination.
src_blk_id	unsigned long	I	m	ID d'item du bloc source.
src_blk_num	unsigned long	I	m	Numéro d'occurrence du bloc source.
src_id	unsigned long	I	m	ID d'item de la variable source.
src_member_id	unsigned long	I	m	ID de membre de la variable source.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.35 Builtin atan**Objet**

Le Builtin atan retourne l'arc-tangente d'une valeur à virgule flottante.

Structure lexicale

atan(x)

Les éléments lexicaux du Builtin atan sont présentés dans le Tableau 38.

Tableau 38 – Builtin atan

Nom du paramètre	Type	Direction	Usage	Description
x	double	I	m	Valeur à virgule flottante.
<return>	double	O	m	Arc-tangente.

4.36 Builtin atof**Objet**

Le Builtin atof convertit un texte en valeur à virgule flottante.

Structure lexicale

atof(str)

Les éléments lexicaux du Builtin atof sont présentés dans le Tableau 39.

Tableau 39 – Builtin atof

Nom du paramètre	Type	Direction	Usage	Description
str	char[]	I	m	Texte à convertir.
<return>	float	O	m	Valeur à virgule flottante.

4.37 Builtin atoi

Objet

Le Builtin convertit un texte en valeur entière.

Structure lexicale

atoi(str)

Les éléments lexicaux du Builtin atoi sont présentés dans le Tableau 40.

Tableau 40 – Builtin atoi

Nom du paramètre	Type	Direction	Usage	Description
str	char[]	I	m	Texte à convertir.
<return>	int	O	m	Valeur entière.

4.38 Builtin browselidentity

Objet

La déclaration BLOB peut ne pas avoir une IDENTITY déclarée. La fonction browselidentity permet à l'utilisateur de sélectionner la déclaration BLOB au moyen d'un mécanisme spécifique à la Host Application (application hôte).

Structure lexicale

browseIdentity(blob_item)

Les éléments lexicaux du Builtin browselidentity sont présentés dans le Tableau 41.

Tableau 41 – Builtin browselidentity

Nom du paramètre	Type	Direction	Usage	Description
blob_item	DD_ITEM &	I	m	BLOB pour laquelle l'identité est parcourue
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau 367.

4.39 Builtin BUILD_MESSAGE

Objet

Le Builtin BUILD_MESSAGE retourne une chaîne basée sur une spécification de format. Toute référence de variable contenue au format chaîne est développée en valeur actuelle. Des variables locales de méthode et des VARIABLES peuvent être référencées dans le paramètre de message. Pour plus d'informations, consulter le Builtin PUT_MESSAGE.

Structure lexicale

BUILD_MESSAGE(string)

Les éléments lexicaux du Builtin BUILD_MESSAGE sont présentés dans le Tableau 42.

Tableau 42 – Builtin BUILD_MESSAGE

Nom du paramètre	Type	Direction	Usage	Description
string	DD_STRING	I	m	Spécification de format.
<return>	DD_STRING	O	m	Chaîne formatée.

4.40 Builtin ByteToDouble

Objet

Le Builtin ByteToDouble retourne une valeur double précision à partir d'un vecteur de huit octets.

Structure lexicale

ByteToDouble(in1, in2, in3, in4, in5, in6, in7, in8)

Les éléments lexicaux du Builtin ByteToDouble sont présentés dans le Tableau 43.

Tableau 43 – Builtin ByteToDouble

Nom du paramètre	Type	Direction	Usage	Description
in1	unsigned char	I	m	Premier élément de la valeur double précision (octet de poids fort).
in2	unsigned char	I	m	Deuxième élément de la valeur double précision.
in3	unsigned char	I	m	Troisième élément de la valeur double précision.
in4	unsigned char	I	m	Quatrième élément de la valeur double précision.
in5	unsigned char	I	m	Cinquième élément de la valeur double précision.
in6	unsigned char	I	m	Sixième élément de la valeur double précision.
in7	unsigned char	I	m	Septième élément de la valeur double précision.
in8	unsigned char	I	m	Huitième élément de la valeur double précision (octet de poids faible).
<return>	double	O	m	Double valeur à partir du vecteur donné de huit octets.

4.41 Builtin ByteToFloat

Objet

Le Builtin ByteToFloat retourne une valeur à virgule flottante à partir d'un vecteur de quatre octets.

Structure lexicale

ByteToFloat(in1, in2, in3, in4)

Les éléments lexicaux du Builtin ByteToFloat sont présentés dans le Tableau 44.

Tableau 44 – Builtin ByteToFloat

Nom du paramètre	Type	Direction	Usage	Description
in1	unsigned char	I	m	Premier élément de la valeur à virgule flottante (octet de poids fort).
in2	unsigned char	I	m	Deuxième élément de la valeur à virgule flottante.
in3	unsigned char	I	m	Troisième élément de la valeur à virgule flottante.
in4	unsigned char	I	m	Quatrième élément de la valeur à virgule flottante (octet de poids faible).
<return>	float	O	m	Valeur à virgule flottante à partir du vecteur donné de quatre octets.

4.42 Builtin ByteToLong

Objet

Le Builtin ByteToLong retourne une valeur longue à partir d'un vecteur de quatre octets.

Structure lexicale

ByteToLong(in1, in2, in3, in4)

Les éléments lexicaux du Builtin ByteToLong sont présentés dans le Tableau 45.

Tableau 45 – Builtin ByteToLong

Nom du paramètre	Type	Direction	Usage	Description
in1	unsigned char	I	m	Premier élément de la valeur longue (octet de poids fort).
in2	unsigned char	I	m	Deuxième élément de la valeur longue.
in3	unsigned char	I	m	Troisième élément de la valeur longue.
in4	unsigned char	I	m	Quatrième élément de la valeur longue (octet de poids faible).
<return>	long	O	m	Valeur longue à partir du vecteur donné de quatre octets.

4.43 Builtin ByteToShort

Objet

Le Builtin ByteToShort retourne une valeur courte à partir d'un vecteur de deux octets.

Structure lexicale

ByteToShort(in1, in2)

Les éléments lexicaux du Builtin ByteToShort sont présentés dans le Tableau 46.

Tableau 46 – Builtin ByteToShort

Nom du paramètre	Type	Direction	Usage	Description
in1	unsigned char	I	m	Premier élément de la valeur courte (octet de poids fort).
in2	unsigned char	I	m	Second élément de la valeur courte (octet de poids faible).
<return>	short	O	m	Valeur courte à partir du vecteur donné de deux octets.

4.44 Builtin cbrt**Objet**

Le Builtin cbrt retourne la racine cubique d'une valeur à virgule flottante.

Structure lexicale

`cbrt(x)`

Les éléments lexicaux du Builtin cbrt sont présentés dans le Tableau 47.

Tableau 47 – Builtin cbrt

Nom du paramètre	Type	Direction	Usage	Description
x	double	I	m	Valeur à virgule flottante.
<return>	double	O	m	Racine cubique.

4.45 Builtin ceil**Objet**

Le Builtin ceil retourne la plus petite valeur entière non inférieure à une valeur à virgule flottante spécifiée.

Structure lexicale

`ceil(x)`

Les éléments lexicaux du Builtin ceil sont présentés dans le Tableau 48.

Tableau 48 – Builtin ceil

Nom du paramètre	Type	Direction	Usage	Description
x	double	I	m	Valeur à virgule flottante.
<return>	double	O	m	Valeur plafond.

4.46 Builtin closeTransferPort**Objet**

La fonction closeTransferPort arrête le transfert de blocs indiqué par le descripteur.

Structure lexicale

`closeTransferPort(handle)`

Les éléments lexicaux du Builtin closeTransferPort sont présentés dans le Tableau 49.

Tableau 49 – Builtin closeTransferPort

Nom du paramètre	Type	Direction	Usage	Description
handle	unsigned long	I	m	Descripteur du transfert de blocs.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau 367.

4.47 Builtin cos**Objet**

Le Builtin cos retourne le cosinus d'une valeur à virgule flottante.

Structure lexicale

`cos (x)`

Les éléments lexicaux du Builtin cos sont présentés dans le Tableau 50.

Tableau 50 – Builtin cos

Nom du paramètre	Type	Direction	Usage	Description
x	double	I	m	Valeur à virgule flottante.
<return>	double	O	m	Cosinus.

4.48 Builtin cosh**Objet**

Le Builtin cosh retourne le cosinus hyperbolique d'une valeur à virgule flottante.

Structure lexicale

`cosh (x)`

Les éléments lexicaux du Builtin cosh sont présentés dans le Tableau 51.

Tableau 51 – Builtin cosh

Nom du paramètre	Type	Direction	Usage	Description
x	double	I	m	Valeur à virgule flottante.
<return>	double	O	m	Cosinus hyperbolique.

4.49 Builtin dassign**Objet**

Le Builtin dassign assigne une valeur à une VARIABLE.

Structure lexicale

`dassign(device_var, new_value)`

Les éléments lexicaux du Builtin dassign sont présentés dans le Tableau 52.

Tableau 52 – Builtin dassign

Nom du paramètre	Type	Direction	Usage	Description
device_var	reference	I	m	Nom de la variable.
new_value	double	I	m	Nouvelle valeur de la variable référencée.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau 367.

4.50 Builtin DATE_AND_TIME_VALUE_to_string

Objet

Le Builtin DATE_AND_TIME_VALUE_to_string crée une représentation de chaîne d'une DATE et une TIME_VALUE(4) (valeur de temps). La fonction de langage de programmation C strftime définit la structure de la chaîne de format.

Structure lexicale

DATE_AND_TIME_VALUE_to_string(string, format, date, time_value)

Les éléments lexicaux du Builtin DATE_AND_TIME_VALUE_to_string sont présentés dans le Tableau 53.

Tableau 53 – Builtin DATE_AND_TIME_VALUE_to_string

Nom du paramètre	Type	Direction	Usage	Description
string	DD_STRING	I/O	m	Chaîne dans laquelle les DATE et TIME_VALUE formatées sont copiées.
format	DD_STRING	I	m	Indique comment les DATE et TIME_VALUE sont à formater.
date	long	I	m	Valeur longue contenant la DATE à formater.
time_value	unsigned long	I	m	TIME_VALUE à formater.
<return>	int	O	m	Nombre de caractères réellement copiés ou -1 en cas d'erreur.

4.51 Builtin Date_to_DayOfMonth

Objet

Le Builtin Date_to_DayOfMonth calcule le jour du mois à partir d'un nombre de jours donnés depuis le 1er janvier 1900.

Structure lexicale

Date_to_DayOfMonth(date)

Les éléments lexicaux du Builtin Date_to_DayOfMonth sont présentés dans le Tableau 54.

Tableau 54 – Builtin Date_to_DayOfMonth

Nom du paramètre	Type	Direction	Usage	Description
date	long	I	m	Valeur longue contenant la DATE.
<return>	int	O	m	Jour du mois.

4.52 Builtin DATE_to_days

Objet

Le Builtin DATE_to_days calcule le nombre de jours entre deux dates.

Structure lexicale

DATE_to_days(date1, date0)

Les éléments lexicaux du Builtin DATE_to_days sont présentés dans le Tableau 55.

Tableau 55 – Builtin DATE_to_days

Nom du paramètre	Type	Direction	Usage	Description
date1	long	I	m	Date la plus récente.
date0	long	I	m	Date la moins récente.
<return>	long	O	m	Nombre de jours entre la date1 et la date0.

4.53 Builtin Date_to_Month

Objet

Le Builtin Date_to_Month calcule le mois à partir d'un nombre donné de jours depuis le 1er janvier 1900.

Structure lexicale

Date_to_Month(date)

Les éléments lexicaux du Builtin Date_to_Month sont présentés dans le Tableau 56.

Tableau 56 – Builtin Date_to_Month

Nom du paramètre	Type	Direction	Usage	Description
date	long	I	m	Longue valeur contenant la DATE.
<return>	int	O	m	Mois.

4.54 Builtin DATE_to_string

Objet

Le Builtin DATE_to_string crée une représentation de chaîne d'une DATE. La fonction de langage de programmation C strftime définit la structure de la chaîne de format.

Structure lexicale

DATE_to_string(string, format, date)

Les éléments lexicaux du Builtin DATE_to_string sont présentés dans le Tableau 57.

Tableau 57 – Builtin DATE_to_string

Nom du paramètre	Type	Direction	Usage	Description
string	DD_STRING	I	m	Chaîne dans laquelle la DATE formatée est copiée.
format	DD_STRING	I	m	Indique comment la DATE est à formater.
date	long	I	m	Valeur longue contenant la DATE.
<return>	int	O	m	Nombre de caractères effectivement copiés ou -1 en cas d'erreur.

4.55 Builtin Date_To_Time**Objet**

Le Builtin Date_To_Time crée un time_t à partir d'une valeur de date. L'heure du jour est définie à 00:00:00.

Structure lexicale

Date_To_Time(date)

Les éléments lexicaux du Builtin Date_To_Time sont présentés dans le Tableau 58.

Tableau 58 – Builtin Date_To_Time

Nom du paramètre	Type	Direction	Usage	Description
date	long	I	m	Valeur longue contenant la DATE
<return>	time_t	O	m	Valeur time_t.

4.56 Builtin Date_to_Year**Objet**

Le Builtin Date_to_Year calcule l'année à partir d'un nombre donné de jours depuis le 1er janvier 1900.

Structure lexicale

Date_to_Year(date)

Les éléments lexicaux du Builtin Date_to_Year sont présentés dans le Tableau 59.

Tableau 59 – Builtin Date_to_Year

Nom du paramètre	Type	Direction	Usage	Description
date	long	I	m	Valeur longue contenant la DATE.
<return>	int	O	m	Année (4 chiffres).

4.57 Builtin days_to_DATE**Objet**

Le Builtin days_to_DATE calcule une date future en ajoutant un nombre de jours à une DATE.

Structure lexicale

days_to_DATE(days, date)

Les éléments lexicaux du Builtin days_to_DATE sont présentés dans le Tableau 60.

Tableau 60 – Builtin days_to_DATE

Nom du paramètre	Type	Direction	Usage	Description
days	long	I	m	Nombre de jours à ajouter à la DATE.
date	long	I	m	Valeur longue contenant la DATE initiale.
<return>	long	O	m	Valeur longue contenant la DATE

4.58 Builtin DELAY

Objet

Le Builtin DELAY affiche un message d'invite et reste en pause pendant le nombre spécifié de secondes. L'invite peut contenir des valeurs de variable locale (voir Builtin put_message pour la structure lexicale). L'invite peut NE PAS contenir des valeurs de variable du dispositif. Le délai doit être un nombre positif.

NOTE 1 Il n'est pas garanti que les messages précédemment affichés restent à l'écran.

NOTE 2 Certaines applications EDD effacent l'affichage à chaque appel de cette fonction.

Structure lexicale

DELAY(delay_time, prompt)

Les éléments lexicaux du Builtin DELAY sont présentés dans le Tableau 61.

Tableau 61 – Builtin DELAY

Nom du paramètre	Type	Direction	Usage	Description
delay_time	unsigned integer	I	m	Délai en nombre de secondes.
prompt	char[]	I	m	Message à afficher.
<return>	VOID	—	—	—

4.59 Builtin delay

Objet

Le Builtin delay affiche un message d'invite, et reste en pause pendant le nombre spécifié de secondes. L'invite peut contenir des valeurs de variable locale et/ou du dispositif (voir put_message pour la structure lexicale). Le délai doit être un nombre positif.

NOTE 1 Il n'est pas garanti que les messages précédemment affichés restent à l'écran.

NOTE 2 Certaines applications EDD effacent l'affichage à chaque appel de cette fonction.

Structure lexicale

delay(delay_time, prompt, global_var_ids)

Les éléments lexicaux du Builtin delay sont présentés dans le Tableau 62.

Tableau 62 – Builtin delay

Nom du paramètre	Type	Direction	Usage	Description
delay_time	unsigned integer	l	m	Délai en nombre de secondes.
prompt	char[]	l	m	Message à afficher.
global_var_ids	array of int	l	m	Matrice contenant des identifiants uniques d'instances de VARIABLE.
<return>	VOID	—	—	—

4.60 Builtin DELAY_TIME**Objet**

Le Builtin DELAY_TIME reste en pause pendant le nombre spécifié de secondes. Le délai doit être un nombre positif.

Structure lexicale

DELAY_TIME(delay_time)

Les éléments lexicaux du Builtin DELAY_TIME sont présentés dans le Tableau 63.

Tableau 63 – Builtin DELAY_TIME

Nom du paramètre	Type	Direction	Usage	Description
delay_time	unsigned integer	l	m	Délai en nombre de secondes.
<return>	VOID	—	—	—

4.61 Builtin delayfor**Objet**

Le Builtin delayfor imprime une chaîne sur l'écran d'application et attend que la durée allouée expire. Des valeurs de VARIABLE peuvent être incorporées dans la chaîne de message avec des informations de formatage.

Structure lexicale

delayfor(duration, prompt, ids, indices, id_count)

Les éléments lexicaux du Builtin delayfor sont présentés dans le Tableau 64.

Tableau 64 – Builtin delayfor

Nom du paramètre	Type	Direction	Usage	Description
duration	long	I	m	Délai en nombre de secondes.
prompt	char[]	I	m	Message à afficher.
ids	array of unsigned long	I	m	ID d'élément d'une instance de VARIABLE utilisée dans le message d'invite.
indices	array of unsigned long	I	m	ID de membre ou index de matrice des variables utilisées dans le message d'invite. Défini à zéro s'il n'y a pas d>ID de membre ou d'index de matrice.
id_count	long	I	m	Nombre d'éléments de matrice dans les matrices pour les ID et les index. Défini à zéro si les matrices pour les ID et les index sont NULL.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.62 Builtin delayfor2

Objet

Le Builtin delayfor2 imprime une chaîne sur l'écran d'application et attend que le temps alloué expire. Des valeurs de VARIABLE peuvent être incorporées dans la chaîne de message avec des informations de formatage.

Structure lexicale

delayfor2(duration, prompt, blk_ids, blk_nums, ids, indices, id_count)

Les éléments lexicaux du Builtin delayfor2 sont présentés dans le Tableau 65.

Tableau 65 – Builtin delayfor2

Nom du paramètre	Type	Direction	Usage	Description
duration	long	I	m	Délai en nombre de secondes.
prompt	char[]	I	m	Message à afficher.
blk_ids	array of unsigned long	I	m	ID d'élément des instances de BLOCK_A utilisées dans le message d'invite.
blk_nums	array of unsigned long	I	m	Numéros d'occurrence des instances de BLOCK_A utilisées dans le message d'invite.
ids	array of unsigned long	I	m	ID d'élément d'une instance de VARIABLE utilisée dans le message d'invite.
indices	array of unsigned long	I	m	ID de membre ou index de matrice des variables utilisées dans le message d'invite. Défini à zéro s'il n'y a pas d>ID de membre ou d'index de matrice.
id_count	long	I	m	Nombre d'éléments de matrice dans les matrices pour les ID et les index. Défini à zéro si les matrices pour les ID et les index sont NULL.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.63 Builtin DICT_ID

Objet

Le Builtin DICT_ID peut être utilisé pour spécifier un ID de dictionnaire de texte pour les appels pour obtenir `stddict_string`. Il convertit un nom de dictionnaire de texte en ID.

Une entrée du dictionnaire de texte est constituée de trois parties: un ID unique, un nom, et une description multilingue. Cette fonction extrait l'ID unique en utilisant le nom du dictionnaire de texte.

Structure lexicale

`DICT_ID(name)`

Les éléments lexicaux du Builtin DICT_ID sont présentés dans le Tableau 66.

Tableau 66 – Builtin DICT_ID

Nom du paramètre	Type	Direction	Usage	Description
<name>	reference	I	m	Nom de la chaîne de dictionnaire.
<return>	long	O	m	ID de dictionnaire pour la chaîne nommée.

4.64 Builtin dictionnaire_string

Objet

Le Builtin `dictionnaire_string` retourne la chaîne du dictionnaire associée au nom donné dans la langue actuelle. Si la chaîne n'est pas disponible dans la langue actuelle, la chaîne en anglais est retournée. Si la chaîne n'est pas trouvée dans le dictionnaire, la fonction retourne "ERROR: Dict String Not Found". Le paramètre `dict_string_name` est le descripteur de la chaîne tel qu'il apparaît dans le dictionnaire.

Structure lexicale

`dictionary_string(dict_string_name)`

Les éléments lexicaux du Builtin `dictionnaire_string` sont présentés dans le Tableau 67.

Tableau 67 – Builtin dictionary_string

Nom du paramètre	Type	Direction	Usage	Description
<code>dict_string_name</code>	reference	I	m	Descripteur de la chaîne tel qu'il apparaît dans le dictionnaire.
<return>	DD_STRING	o	m	Chaîne de dictionnaire.

4.65 Builtin DiffTime

Objet

Le Builtin `DiffTime` calcule la différence entre deux heures calendaires.

Structure lexicale

`DiffTime(t1, t0)`

Les éléments lexicaux du Builtin `DiffTime` sont présentés dans le Tableau 68.

Tableau 68 – Builtin DiffTime

Nom du paramètre	Type	Direction	Usage	Description
t1	time_t	l	m	Heure la plus récente.
t0	time_t	l	m	Heure la moins récente.
<return>	double	o	m	Différence entre t1 et t0 en secondes.

4.66 Builtin discard_on_exit

Objet

Le Builtin discard_on_exit (rejeter en sortie) définit l'action finale pour rejeter des changements effectués par la méthode sur des valeurs de VARIABLE stockées en mémoire cache et qui n'ont pas déjà été envoyées au dispositif. Cela conduit la méthode à rejeter les changements apportés aux valeurs de VARIABLE en mémoire cache lorsque la méthode se termine.

Lorsqu'une méthode modifie une valeur de VARIABLE dans une application EDD en mémoire cache, elle modifie une copie résidente de cette valeur. Cette modification n'est effective que pendant la durée de vie de la méthode. Pour modifier la/les valeur(s) dans le dispositif, la méthode doit envoyer la/les valeur(s) au dispositif ou appeler send_on_exit. Pour enregistrer les valeurs uniquement dans la mémoire cache (au lieu de les envoyer au dispositif), appeler le Builtin save_on_exit (enregistrer en sortie).

Après la terminaison de la méthode, les valeurs modifiées dans la mémoire cache reprennent les valeurs du dispositif qui existaient avant qu'elles soient modifiées par la méthode.

Toutes les valeurs qui ont été modifiées mais qui n'ont pas été envoyées au dispositif sont traitées en fin de méthode soit en les rejetant, soit en les envoyant au dispositif, ou en enregistrant les modifications d'une manière persistante (mais qui n'affecte pas les valeurs dans le dispositif).

Ce Builtin n'effectue aucune action dans une application EDD non en mémoire-cache, car les valeurs de paramètre sont écrites directement dans le dispositif.

Si une méthode n'appelle pas discard_on_exit, save_on_exit, ou send_on_exit avant de quitter, les valeurs modifiées par le Builtin sont rejetées comme si le Builtin discard_on_exit avait été appelé.

Structure lexicale

discard_on_exit(VOID)

L'élément lexical du Builtin discard_on_exit est présenté dans le Tableau 69.

Tableau 69 – Builtin discard_on_exit

Nom du paramètre	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.67 Builtin DISPLAY

Objet

Le Builtin DISPLAY (afficher) affiche le message spécifié à l'écran, en mettant à jour en continu les valeurs de variable dynamiques utilisées dans la chaîne (voir PUT_MESSAGE

pour la syntaxe). La valeur de variable continue d'être mise à jour jusqu'à ce que l'opérateur effectue l'accusé de réception.

Structure lexicale

DISPLAY(prompt)

Les éléments lexicaux du Builtin DISPLAY sont présentés dans le Tableau 70.

Tableau 70 – Builtin DISPLAY

Nom du paramètre	Type	Direction	Usage	Description
prompt	DD_STRING	I	m	Message à afficher.
<return>	VOID	—	—	—

4.68 Builtin display

Objet

Le Builtin display affiche le message spécifié à l'écran, en mettant à jour en continu les valeurs de VARIABLE dynamiques utilisées dans la chaîne (voir PUT_MESSAGE pour la structure lexicale). La valeur de VARIABLE continue d'être mise à jour jusqu'à ce que l'opérateur effectue l'accusé de réception.

NOTE 1 Il n'est pas garanti que les messages précédemment affichés restent à l'écran.

NOTE 2 Certaines applications EDD effacent l'affichage à chaque appel de cette fonction.

Structure lexicale

display(prompt, global_var_ids)

Les éléments lexicaux du Builtin display sont présentés dans le Tableau 71.

Tableau 71 – Builtin display

Nom du paramètre	Type	Direction	Usage	Description
prompt	char[]	I	m	Message à afficher.
global_var_ids	array of int	I	m	Matrice contenant des identifiants uniques d'instances de VARIABLE.
<return>	VOID	—	—	—

4.69 Builtin display_bitenum

Objet

Le Builtin display_bitenum affiche une représentation textuelle d'une VARIABLE BIT_ENUMERATED et attend l'accusé de réception de l'utilisateur.

NOTE 1 Il n'est pas garanti que les messages précédemment affichés restent à l'écran.

NOTE 2 Certaines applications EDD effacent l'affichage à chaque appel de cette fonction.

Structure lexicale

display_bitenum(bitenum_variable)

Les éléments lexicaux du Builtin display_bitenum sont présentés dans le Tableau 72.

Tableau 72 – Builtin display_bitenum

Nom du paramètre	Type	Direction	Usage	Description
bitenum_variable	Referenc e	l	m	VARIABLE BIT_ENUMERATED qui doit être affichée.
<return>	VOID	—	—	—

4.70 Builtin display_builtin_error

Objet

Le Builtin display_builtin_error affiche sur le dispositif d'affichage de l'application le texte associé à un Builtin error (erreur) spécifié. Le Builtin display ne retourne pas le contrôle à la méthode jusqu'à ce que l'utilisateur accuse réception du message.

Ce Builtin constitue un moyen de débogage des appels de sous-programmes Builtin qui retournent une erreur. Il convient de ne jamais l'utiliser dans une méthode de production finale, car il convient que les types des erreurs retournées par les Builtin ne soient jamais reçus dans une méthode de débogage total.

Les codes d'erreur de Builtin peuvent être définis dans un fichier séparé.

Structure lexicale

display_builtin_error(error)

Les éléments lexicaux du Builtin display_builtin_error sont présentés dans le Tableau 73.

Tableau 73 – Builtin display_builtin_error

Nom du paramètre	Type	Direction	Usage	Description
error	long	l	m	Code pour une erreur de Builtin.
<return>	long	o	m	Un des codes de retour spécifiés dans le Tableau 366.

4.71 Builtin display_comm_error

Objet

Le Builtin display_comm_error affiche la chaîne de texte associée à une erreur de communication sur le dispositif d'affichage de l'application.

NOTE 1 Une erreur de communication de zéro n'est pas considérée comme une erreur de communication.

NOTE 2 Le Builtin ne retourne pas le contrôle à la méthode jusqu'à ce que l'utilisateur accuse réception du message.

En interne, ce Builtin appelle le Builtin get_comm_error_string pour obtenir la chaîne de texte associée et le Builtin get_acknowledgement pour afficher la chaîne de texte associée et attendre l'accusé de réception de l'utilisateur.

Les codes d'erreur de communication sont définis par les consortiums. Leurs chaînes associées peuvent être définies dans un dictionnaire de texte.

Structure lexicale

display_comm_error(error)

Les éléments lexicaux du Builtin display_comm_error sont présentés dans le Tableau 74.

Tableau 74 – Builtin display_comm_error

Nom du paramètre	Type	Direction	Usage	Description
error	unsigned long	I	m	Code pour une erreur de communication.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.72 Builtin display_comm_status**Objet**

Le Builtin display_comm_status affiche la chaîne associée à la valeur spécifiée de l'octet comm_status_value. Le message reste à l'écran. La chaîne affichée reste à l'écran jusqu'à ce que l'opérateur effectue l'accusé de réception.

NOTE 1 Il n'est pas garanti que les messages précédemment affichés restent à l'écran.

NOTE 2 Certaines applications EDD effacent l'affichage à chaque appel de cette fonction.

Structure lexicale

display_comm_status(comm_status_value)

Les éléments lexicaux du Builtin display_comm_status sont présentés dans le Tableau 75.

Tableau 75 – Builtin display_comm_status

Nom du paramètre	Type	Direction	Usage	Description
comm_status_value	int	I	m	Code pour une erreur de communication.
<return>	VOID	—	—	—

4.73 Builtin display_device_status**Objet**

Le Builtin display_device_status affiche la chaîne associée à la valeur spécifiée de l'octet device_status. La chaîne affichée reste à l'écran jusqu'à ce que l'opérateur effectue l'accusé de réception.

NOTE 1 Il n'est pas garanti que les messages précédemment affichés restent à l'écran.

NOTE 2 Certaines applications EDD effacent l'affichage à chaque appel de cette fonction.

Structure lexicale

display_device_status(device_status_value)

Les éléments lexicaux du Builtin display_device_status sont présentés dans le Tableau 76.

Tableau 76 – Builtin display_device_status

Nom du paramètre	Type	Direction	Usage	Description
device_status_value	int	I	m	Code pour le statut du dispositif.
<return>	VOID	—	—	—

4.74 Builtin display_dynamics

Objet

Le Builtin display_dynamics affiche un message en continu sur le dispositif de sortie de l'application. Le Builtin met à jour les valeurs spécifiées dans le message jusqu'à ce l'utilisateur effectue l'accusé de réception.

Des valeurs de VARIABLE peuvent être incorporées dans la chaîne de message avec des informations de formatage. Cet appel peut être effectué pour des VARIABLES du dispositif ou des VARIABLES de méthode locales. Le principal objectif de l'appel est d'afficher des informations de valeur du dispositif que l'utilisateur souhaite examiner pendant une période.

Structure lexicale

display_dynamics(prompt, ids, indices, id_count)

Les éléments lexicaux du Builtin display_dynamics sont présentés dans le Tableau 77.

Tableau 77 – Builtin display_dynamics

Nom du paramètre	Type	Direction	Usage	Description
prompt	char[]	I	m	Message à afficher.
ids	array of unsigned long	I	m	ID d'élément des VARIABLES utilisées dans le message d'invite.
indices	array of unsigned long	I	m	ID de membre ou index de matrice des VARIABLES utilisées dans le message d'invite. Défini à zéro s'il n'y a pas d>ID de membre ou d'index de matrice.
id_count	long	I	m	Nombre d'éléments de matrice dans les matrices pour les ID et les index. Défini à zéro si les matrices pour les ID et les index sont NULL.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366. ID

4.75 Builtin display_dynamics2

Objet

Le Builtin display_dynamics2 affiche un message en continu sur le dispositif de sortie de l'application. Le Builtin met à jour les valeurs spécifiées dans le message jusqu'à ce qu'il soit interrompu par l'utilisateur. Jusqu'à ce que l'utilisateur l'interrompe, le contrôle n'est pas retourné à la méthode. (Le code d'application recherche l'accusé de réception; le Builtin attend que le code d'application reçoive une intervention de l'utilisateur, la vérifie, et retourne contrôle au Builtin.)

Des valeurs de VARIABLE peuvent être incorporées dans la chaîne de message avec des informations de formatage. Cet appel peut être effectué pour des VARIABLES du dispositif ou des VARIABLES de méthode locales. Le principal objectif de l'appel est d'afficher des informations de valeur du dispositif que l'utilisateur souhaite examiner pendant une période.

Structure lexicale

display_dynamics2(prompt, blk_ids, blk_nums, ids, indices, id_count)

Les éléments lexicaux du Builtin display_dynamics2 sont présentés dans le Tableau 78.

Tableau 78 – Builtin display_dynamics2

Nom du paramètre	Type	Direction	Usage	Description
prompt	char[]	I	m	Message à afficher.
blk_ids	array of unsigned long	I	m	ID d'élément des instances de BLOCK_A utilisées dans le message d'invite.
blk_nums	array of unsigned long	I	m	Numéros d'occurrence des instances de BLOCK_A utilisées dans le message d'invite.
ids	array of unsigned long	I	m	ID d'élément des VARIABLES utilisées dans le message d'invite.
indices	array of unsigned long	I	m	ID de membre ou index de matrice des VARIABLES utilisées dans le message d'invite. Défini à zéro s'il n'y a pas d>ID de membre ou d'index de matrice.
id_count	long	I	m	Nombre d'éléments de matrice dans les matrices pour les ID et les index. Défini à zéro si les matrices pour les ID et les index sont NULL.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366

4.76 Builtin display_message

Objet

Le Builtin display_message affiche un message sur le dispositif de sortie d'application. Il retourne le contrôle à la méthode sans nécessiter un accusé de réception de l'utilisateur.

Les valeurs de VARIABLE de méthode locale du dispositif et les étiquettes pour les VARIABLE du dispositif peuvent être incorporées dans la chaîne d'invite avec des informations de formatage. Cet appel est utilisé pour informer l'utilisateur de changements de statut du système qui ne requièrent aucune action.

Structure lexicale

```
display_message(prompt, ids, indices, id_count)
```

Les éléments lexicaux du Builtin display_message sont présentés dans le Tableau 79.

Tableau 79 – Builtin display_message

Nom du paramètre	Type	Direction	Usage	Description
prompt	char[]	I	m	Message à afficher.
ids	array of unsigned long	I	m	ID d'élément des VARIABLES utilisées dans le message d'invite.
indices	array of unsigned long	I	m	ID de membre ou index de matrice des VARIABLES utilisées dans le message d'invite. Défini à zéro s'il n'y a pas d>ID de membre ou d'index de matrice.
id_count	long	I	m	Nombre d'éléments de matrice dans les matrices pour les ID et les index. Défini à zéro si les matrices pour les ID et les index sont NULL.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.77 Builtin display_message2

Objet

Le Builtin display_message2 affiche un message sur le dispositif de sortie d'application. Il retourne le contrôle à la méthode sans nécessiter un accusé de réception de l'utilisateur.

Les valeurs de VARIABLE de méthode locale du dispositif et les étiquettes pour les VARIABLE du dispositif peuvent être incorporées dans la chaîne d'invite avec des informations de formatage. Cet appel est utilisé pour informer l'utilisateur de changements de statut du système qui ne requièrent aucune action.

Structure lexicale

display_message2(prompt, blk_ids, blk_nums, ids, indices, id_count)

Les éléments lexicaux du Builtin display_message2 sont présentés dans le Tableau 80.

Tableau 80 – Builtin display_message2

Nom du paramètre	Type	Direction	Usage	Description
prompt	char[]	I	m	Message à afficher.
blk_ids	array of unsigned long	I	m	ID d'élément des instances de BLOCK_A utilisées dans le message d'invite.
blk_nums	array of unsigned long	I	m	Numéros d'occurrence des instances de BLOCK_A utilisées dans le message d'invite.
ids	array of unsigned long	I	m	ID d'élément des VARIABLES utilisées dans le message d'invite.
indices	array of unsigned long	I	m	ID de membre ou index de matrice des VARIABLES utilisées dans le message d'invite. Défini à zéro s'il n'y a pas d>ID de membre ou d'index de matrice.
id_count	long	I	m	Nombre d'éléments de matrice dans les matrices pour les ID et les index. Défini à zéro si les matrices pour les ID et les index sont NULL.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.78 Builtin display_response_code

Objet

Le Builtin display_response_code affiche la description d'un code de réponse pour un élément. Le Builtin display ne retourne pas le contrôle à la méthode jusqu'à ce que l'utilisateur accuse réception du message.

En interne, ce Builtin appelle le Builtin get_response_code_string pour obtenir la chaîne de texte associée et le Builtin get_acknowledgement pour afficher la chaîne de texte associée et attendre l'accusé de réception de l'utilisateur.

Un code de réponse est une valeur représentant une condition d'erreur spécifique à une application. Une valeur de code de réponse de zéro n'est pas considérée comme un code de réponse d'erreur. Chaque code de réponse et type de code de réponse est défini dans l'EDD.

L>ID de paramètre et le member_id désignent un paramètre, qui peut avoir un code de réponse associé à celui-ci après avoir été lu ou écrit. L>ID de paramètre et le member_id sont spécifiés en utilisant les Builtin ITEM_ID et MEMBER_ID et le nom ou la référence de la VARIABLE en question.

Structure lexicale

```
display_response_code(id, member_id, code)
```

Les éléments lexicaux du Builtin `display_response_code` sont présentés dans le Tableau 81.

Tableau 81 – Builtin `display_response_code`

Nom du paramètre	Type	Direction	Usage	Description
ID	unsigned long	I	m	ID d'élément de l'élément qui a généré le code de réponse.
member_id	unsigned long	I	m	ID de membre de l'élément qui a généré le code de réponse.
code	unsigned long	I	m	Code de réponse reçu depuis le Builtin <code>get_response_code</code> .
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.79 Builtin `display_response_status`**Objet**

Le Builtin `display_response_status` affiche la chaîne associée à la valeur spécifiée de l'octet `response_code`. La chaîne affichée reste à l'écran jusqu'à ce que l'opérateur effectue l'accusé de réception.

NOTE 1 Il n'est pas garanti que les messages précédemment affichés restent à l'écran.

NOTE 2 Certaines applications EDD effacent l'affichage à chaque appel de cette fonction.

Structure lexicale

```
display_response_status(response_code_value)
```

Les éléments lexicaux du Builtin `display_response_status` sont présentés dans le Tableau 82.

Tableau 82 – Builtin `display_response_status`

Nom du paramètre	Type	Direction	Usage	Description
Command_name	int	I	M	NUMBER de la COMMAND ayant envoyé le statut de réponse
response_code_value	int	I	m	Code de réponse à afficher dans une invite.
<return>	VOID	—	—	—

4.80 Builtin `display_xmtr_status`**Objet**

Le Builtin `display_xmtr_status` affiche la chaîne associée à la valeur spécifiée de la VARIABLE émettrice indiquée. La chaîne affichée reste à l'écran jusqu'à ce que l'opérateur effectue l'accusé de réception.

NOTE 1 Il n'est pas garanti que les messages précédemment affichés restent à l'écran.

NOTE 2 Certaines applications EDD effacent l'affichage à chaque appel de cette fonction.

Structure lexicale

```
display_xmtr_status(xmtr_variable, response_code_value)
```

Les éléments lexicaux du Builtin `display_xmtr_status` sont présentés dans le Tableau 83.

Tableau 83 – Builtin display_xmtr_status

Nom du paramètre	Type	Direction	Usage	Description
xmtr_variable	int	I	m	Référence à une instance de VARIABLE.
response_code_value	int	I	m	Valeur de code de réponse.
<return>	VOID	—	—	—

4.81 Builtin DoubleToByte

Objet

Le Builtin DoubleToByte fournit un vecteur de valeurs de caractère non signé représentant les éléments particuliers d'une valeur double précision.

Structure lexicale

DoubleToByte(in, out1, out2, out3, out4, out5, out6, out7, out8)

Les éléments lexicaux du Builtin DoubleToByte sont présentés dans le Tableau 84.

Tableau 84 – Builtin DoubleToByte

Nom du paramètre	Type	Direction	Usage	Description
in	double	I	m	Valeur double précision à convertir.
out1	unsigned char	O	m	Premier élément de la valeur double précision (octet de poids fort).
out2	unsigned char	O	m	Deuxième élément de la valeur double précision.
out3	unsigned char	O	m	Troisième élément de la valeur double précision.
out4	unsigned char	O	m	Quatrième élément de la valeur double précision.
out5	unsigned char	O	m	Cinquième élément de la valeur double précision.
out6	unsigned char	O	m	Sixième élément de la valeur double précision.
out7	unsigned char	O	m	Septième élément de la valeur double précision.
out8	unsigned char	O	m	Huitième élément de la valeur double précision (octet de poids faible).
<return>	VOID	—	—	—

4.82 Builtin drand

Objet

Le Builtin drand() retourne les valeurs double précision aléatoires réparties uniformément sur la plage comprise entre 0 et 1.

Structure lexicale

double drand()

L'élément lexical du Builtin drand est présenté dans le Tableau 85.

Tableau 85 – Builtin drand

Nom du paramètre	Type	Direction	Usage	Description
<return>	double	O	m	Retourne les valeurs double précision aléatoires réparties uniformément sur la plage comprise entre 0 et 1.

4.83 Builtin dseed**Objet**

Le Builtin dseed() prend une valeur double précision dans la plage comprise entre 0 et 1 comme valeur d'origine pour le générateur de nombres pseudoaléatoires. Pour une mise en œuvre hôte donnée, si le Builtin dseed est appelé avec la même valeur d'origine avec laquelle la même séquence de nombres pseudoaléatoires doit être générée.

Structure lexicale

```
void dseed(double seed)
```

Les éléments lexicaux du Builtin dseed sont présentés dans le Tableau 86.

Tableau 86 – Builtin dseed

Nom du paramètre	Type	Direction	Usage	Description
seed	double	I	m	Valeur double précision dans la plage comprise entre 0 et 1 utilisée comme valeur d'origine du générateur de nombres pseudoaléatoires.
<return>	VOID	—	—	—

4.84 Builtin edit_device_value**Objet**

Le Builtin edit_device_value affiche une invite avec une valeur du dispositif spécifiée par le param_id et le param_member_id. L'utilisateur peut éditer la valeur affichée.

Une valeur du dispositif peut résider dans la mémoire cache d'une application EDD en mémoire cache, et la valeur peut ne pas être écrite dans le dispositif jusqu'à ce qu'une méthode soit terminée ou la valeur est explicitement envoyée au dispositif par la méthode.

Des valeurs de VARIABLE peuvent être incorporées dans la chaîne de message avec des informations de formatage. L'application édite et vérifie les valeurs entrées pour la valeur du dispositif. Cette édition est basée sur des types d'arithmétique de VARIABLE définis dans l'EDD originale pour chaque VARIABLE. Les attributs EDDL associés aux VARIABLES peuvent comprendre

- HANDLING
- DISPLAY_FORMAT
- EDIT_FORMAT
- MIN_VALUE
- MAX_VALUE

Structure lexicale

```
edit_device_value(prompt, ids, indices, id_count, param_id,
param_member_id)
```

Les éléments lexicaux du Builtin edit_device_value sont présentés dans le Tableau 87.

Tableau 87 – Builtin edit_device_value

Nom du paramètre	Type	Direction	Usage	Description
prompt	char[]	I	m	Message à afficher.
ids	array of unsigned long	I	m	ID d'élément des VARIABLES utilisées dans le message d'invite.
indices	array of unsigned long	I	m	ID de membre ou index de matrice des VARIABLES utilisées dans le message d'invite. Défini à zéro s'il n'y a pas d>ID de membre ou d'index de matrice.
id_count	long	I	m	Nombre d'éléments de matrice dans les matrices pour les ID et les index. Défini à zéro si les matrices pour les ID et les index sont NULL.
param_id	unsigned long	I	m	ID d'élément de la VARIABLE à modifier.
param_member_id	unsigned long	I	m	ID de membre de la VARIABLE à modifier.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.85 Builtin edit_device_value2

Objet

Le Builtin edit_device_value2 affiche une invite avec une valeur du dispositif spécifiée par le param_id et le param_member_id. L'utilisateur peut éditer la valeur affichée.

Une valeur du dispositif peut résider dans la mémoire cache d'une application EDD en mémoire cache, et la valeur peut ne pas être écrite dans le dispositif jusqu'à ce qu'une méthode soit terminée ou la valeur est explicitement envoyée au dispositif par la méthode.

Des valeurs de VARIABLE peuvent être incorporées dans la chaîne de message avec des informations de formatage. L'application édite et vérifie les valeurs entrées pour la valeur du dispositif. Cette édition est basée sur des types d'arithmétique de VARIABLE définis dans l'EDD originale pour chaque VARIABLE. Les attributs EDDL associés aux VARIABLES peuvent comprendre:

- HANDLING
- DISPLAY_FORMAT
- EDIT_FORMAT
- MIN_VALUE
- MAX_VALUE

Structure lexicale

```
edit_device_value2(prompt, blk_ids, blk_nums, ids, indices, id_count,
blk_id, blk_num, param_id, param_member_id)
```

Les éléments lexicaux du Builtin ACKNOWLEDGE sont présentés dans le Tableau 88.

Tableau 88 – Builtin edit_device_value2

Nom du paramètre	Type	Direction	Usage	Description
prompt	char[]	l	m	Message à afficher.
blk_ids	array of unsigned long	l	m	ID d'élément des instances de BLOCK_A utilisées dans le message d'invite.
blk_nums	array of unsigned long	l	m	Numéros d'occurrence des instances de BLOCK_A utilisées dans le message d'invite.
ids	array of unsigned long	l	m	ID d'élément des VARIABLES utilisées dans le message d'invite.
indices	array of unsigned long	l	m	ID de membre ou index de matrice des VARIABLES utilisées dans le message d'invite. Défini à zéro s'il n'y a pas d>ID de membre ou d'index de matrice.
id_count	long	l	m	Nombre d'éléments de matrice dans les matrices pour les ID et les index. Défini à zéro si les matrices pour les ID et les index sont NULL.
blk_id	unsigned long	l	m	ID d'élément de l'instance de BLOCK_A à modifier.
blk_num	unsigned long	l	m	Numéro d'occurrence de l'instance de BLOCK_A à modifier.
param_id	unsigned long	l	m	ID d'élément de la VARIABLE à modifier.
param_member_id	unsigned long	l	m	ID de membre de la VARIABLE à modifier.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366

4.86 Builtin edit_local_value

Objet

Le Builtin edit_local_value affiche une invite avec le nom de VARIABLE de méthode locale spécifié par le paramètre local_var. Le paramètre local_var est une chaîne de caractères C à terminaison null. L'utilisateur peut éditer la valeur affichée de la VARIABLE de la méthode locale.

Les valeurs de variable peuvent être incorporées dans la chaîne de message avec des informations de formatage. L'application EDD est responsable de la prévention du retour de valeurs non valides en effectuant des vérifications basées sur le type de VARIABLE uniquement. La méthode doit effectuer une vérification d'intervalle sur les valeurs retournées.

Structure lexicale

```
edit_local_value(prompt, ids, indices, id_count, local_var)
```

Les éléments lexicaux du Builtin edit_local_value sont présentés dans le Tableau 89.

Tableau 89 – Builtin edit_local_value

Nom du paramètre	Type	Direction	Usage	Description
prompt	char[]	I	m	Message à afficher.
ids	array of unsigned long	I	m	ID d'élément des VARIABLES utilisées dans le message d'invite.
indices	array of unsigned long	I	m	ID de membre ou index de matrice des VARIABLES utilisées dans le message d'invite. Défini à zéro s'il n'y a pas d'ID de membre ou d'index de matrice.
id_count	long	I	m	Nombre d'éléments de matrice dans les matrices pour les ID et les index. Défini à zéro si les matrices pour les ID et les index sont NULL.
local_var	char[]	I	m	Nom de la VARIABLE locale qui peut être modifiée.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.87 Builtin edit_local_value2**Objet**

Le Builtin edit_local_value2 affiche une invite avec le nom de VARIABLE de méthode locale spécifié par le paramètre local_var. Le paramètre local_var est une chaîne de caractères C à terminaison null. L'utilisateur peut éditer la valeur affichée de la VARIABLE de la méthode locale.

Les valeurs de variable peuvent être incorporées dans la chaîne de message avec des informations de formatage. L'application EDD est responsable de la prévention du retour de valeurs non valides en effectuant des vérifications basées sur le type de VARIABLE uniquement. La méthode doit effectuer une vérification d'intervalle sur les valeurs retournées.

Structure lexicale

```
edit_local_value2(prompt, blk_ids, blk_nums, ids, indices, id_count, local_var)
```

Les éléments lexicaux du Builtin edit_local_value2 sont présentés dans le Tableau 90.

Tableau 90 – Builtin edit_local_value2

Nom du paramètre	Type	Direction	Usage	Description
prompt	char[]	I	m	Message à afficher.
blk_ids	array of unsigned long	I	m	ID d'élément des instances de BLOCK_A utilisées dans le message d'invite.
blk_nums	array of unsigned long	I	m	Numéros d'occurrence des instances de BLOCK_A utilisées dans le message d'invite.
ids	array of unsigned long	I	m	ID d'élément des VARIABLES utilisées dans le message d'invite.
indices	array of unsigned long	I	m	ID de membre ou index de matrice des VARIABLES utilisées dans le message d'invite. Défini à zéro s'il n'y a pas d>ID de membre ou d'index de matrice.
id_count	long	I	m	Nombre d'éléments de matrice dans les matrices pour les ID et les index. Défini à zéro si les matrices pour les ID et les index sont NULL.
local_var	char[]	I	m	Nom de la VARIABLE locale qui peut être modifiée.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366

4.88 Builtin exp

Objet

Le Builtin exp retourne l'exponentielle d'une valeur à virgule flottante.

Structure lexicale

exp (x)

Les éléments lexicaux du Builtin exp sont présentés dans le Tableau 91.

Tableau 91 – Builtin exp

Nom du paramètre	Type	Direction	Usage	Description
x	double	I	m	Valeur à virgule flottante.
<return>	double	O	m	Valeur exponentielle.

4.89 Builtin ext_send_command

Objet

Le Builtin ext_send_command a la même fonctionnalité que la fonction send_command, sauf que le statut de commande et les octets de données retournés par le dispositif avec la commande 48 sont également retournés.

La fonction appelante est responsable de l'allocation des matrices pour les données à retourner. Les variables cmd_status et more_data_status sont des pointeurs vers des matrices de 3 octets, et la variable more_data_info est un pointeur vers une matrice dont la taille est égale au nombre d'octets de données qui peuvent être retournés par le dispositif dans la commande 48 (maximum 25). Si le bit de statut indiquant que des données supplémentaires sont disponibles est retourné (classe de statut MORE), une commande 48 est automatiquement émise. Si la commande 48 n'a pas été émise, les octets more_data_status et more_data_info octets sont définis à zéro.

Structure lexicale

`ext_send_command(cmd_number, cmd_status, more_data_status, more_data_info)`

Les éléments lexicaux du Builtin `ext_send_command` sont présentés dans le Tableau 92.

Tableau 92 – Builtin `ext_send_command`

Nom du paramètre	Type	Direction	Usage	Description
<code>cmd_number</code>	int	I	m	Numéro de COMMAND.
<code>cmd_status</code>	char[]	I	m	Statut de COMMAND.
<code>more_data_status</code>	char[]	I	m	Spécifie si des données de statut additionnelles sont disponibles.
<code>more_data_info</code>	char[]	I	m	Spécifie si des informations additionnelles sont disponibles.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau 367.

4.90 Builtin `ext_send_command_trans`

Objet

Le Builtin `ext_send_command_trans` envoie la commande avec la transaction spécifiée au dispositif. Cette fonction est destinée à être utilisée pour envoyer des commandes qui ont été définies avec des transactions multiples.

Le Builtin `ext_send_command_trans` a la même fonctionnalité que la fonction `send_command`, sauf que le statut de commande et les octets de données retournés par le dispositif avec la commande 48 sont également retournés.

La fonction appelante est responsable de l'allocation des matrices pour les données à retourner. Les variables `cmd_status` et `more_data_status` sont des pointeurs vers des matrices de 3 octets, et la variable `more_data_info` est un pointeur vers une matrice dont la taille est égale au nombre d'octets de données qui peuvent être retournés par le dispositif dans la commande 48 (maximum 25). Si la commande 48 n'a pas été émise, les octets `more_data_status` et `more_data_info` octets sont définis à zéro.

Structure lexicale

`ext_send_command_trans(cmd_number, transaction, cmd_status, more_data_status, more_data_info)`

Les éléments lexicaux du Builtin `ext_send_command_trans` sont présentés dans le Tableau 93.

Tableau 93 – Builtin `ext_send_command_trans`

Nom du paramètre	Type	Direction	Usage	Description
<code>cmd_number</code>	int	I	m	Numéro de COMMAND.
<code>transaction</code>	int	I	m	Numéro de la TRANSACTION.
<code>cmd_status</code>	char[]	I	m	Statut de COMMAND.
<code>more_data_status</code>	char[]	I	m	Spécifie si des données de statut additionnelles sont disponibles.
<code>more_data_info</code>	char[]	I	m	Spécifie si des informations additionnelles sont disponibles.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau 367.

4.91 Builtin fail_on_all_comm_errors

Objet

Le Builtin fail_on_all_comm_errors indique aux Builtin d'envoyer un code de retour à la méthode au lieu d'interrompre la méthode ou de relancer la demande de communication lorsqu'une erreur de communication est reçue.

Au lieu d'interrompre la méthode ou de tenter de relancer une demande de communication en échec lorsqu'une erreur de communication se produit, le Builtin fail_on_all_comm_errors initialise les Builtin pour envoyer un code d'erreur de communication à la méthode.

Lorsqu'une erreur de communication se produit, le Builtin effectuant une demande de communication à la fois retourne l'erreur BLTIN_FAIL_COMM_ERROR. La méthode peut appeler get_comm_error et ensuite get_comm_error_string pour obtenir des informations additionnelles concernant l'erreur puis afficher la chaîne d'erreur, le cas échéant.

Structure lexicale

fail_on_all_comm_errors (VOID)

L'élément lexical du Builtin fail_on_all_comm_errors est présenté dans le Tableau 94.

Tableau 94 – Builtin fail_on_all_comm_errors

Nom du paramètre	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.92 Builtin fail_on_all_response_codes

Objet

Le Builtin fail_on_all_response_codes indique aux Builtin de retourner un code d'erreur de Builtin à la méthode au lieu d'interrompre la méthode ou de relancer lorsqu'un code de réponse est reçu.

Au lieu d'interrompre la méthode ou de tenter de relancer la demande d'application en échec, le Builtin fail_on_all_response_codes initialise les Builtin pour envoyer un code de retour à la méthode lorsqu'un code de réponse est reçu.

Lorsqu'une erreur de code de réponse se produit, un Builtin quelconque effectuant une demande à la fois retourne l'erreur BLTIN_FAIL_RESPONSE_CODE. La méthode peut appeler get_comm_error et ensuite get_comm_error_string pour obtenir des informations additionnelles concernant l'erreur, le cas échéant.

Un code réponse est une valeur représentant une condition d'erreur spécifique à une application.

NOTE Une valeur de code de réponse zéro n'est pas considérée comme un code de réponse d'erreur. Chaque code de réponse et type de code de réponse est défini dans l'EDD.

Structure lexicale

fail_on_all_response_codes (VOID)

L'élément lexical du Builtin fail_on_all_response_codes est présenté dans le Tableau 95.

Tableau 95 – Builtin fail_on_all_response_codes

Nom du paramètre	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.93 Builtin fail_on_comm_error

Objet

Le Builtin fail_on_comm_error indique aux Builtin de retourner un code de retour à la méthode au lieu d'interrompre la méthode ou de relancer lorsqu'une erreur de communication est reçue.

Au lieu d'interrompre la méthode ou de tenter de relancer une demande de communication en échec, le Builtin fail_on_comm_error initialise les Builtin pour retourner un code d'erreur de Builtin à la méthode lorsqu'une erreur de communication se produit.

Lorsque l'erreur de communication spécifiée se produit, un Builtin quelconque effectuant une demande de communication à la fois retourne l'erreur Builtin fail_on_comm_error. La méthode peut appeler get_comm_error puis get_comm_error_string pour obtenir des informations additionnelles concernant l'erreur et ensuite les afficher, le cas échéant.

Les codes d'erreur de communication sont définis par les consortiums. Leurs chaînes associées peuvent être définies dans un dictionnaire de texte. Une erreur de communication zéro n'est pas considérée comme une erreur de communication.

Structure lexicale

fail_on_comm_error(error)

Les éléments lexicaux du Builtin fail_on_comm_error sont présentés dans le Tableau 96.

Tableau 96 – Builtin fail_on_comm_error

Nom du paramètre	Type	Direction	Usage	Description
error	unsigned long	I	m	Code d'erreur de communication.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.94 Builtin fail_on_response_code

Objet

Le Builtin fail_on_response_code définit les Builtin pour retourner un code d'erreur de Builtin à la méthode au lieu d'interrompre la méthode ou de relancer lorsqu'un code de réponse d'erreur spécifié est reçu.

Au lieu d'interrompre la méthode ou de relancer une demande en échec, le Builtin fail_on_response_code initialise les Builtin pour retourner un code de retour à la méthode lorsqu'une erreur de code de réponse se produit.

Lorsque l'erreur de code de réponse se produit, un Builtin quelconque effectuant une demande à la fois retourne l'erreur BLTIN_FAIL_RESPONSE_CODE. La méthode peut appeler get_comm_error puis get_comm_error_string pour obtenir des informations additionnelles concernant l'erreur et ensuite le cas échéant, les afficher.

Un code de réponse est une valeur représentant une condition d'erreur spécifique à une application.

NOTE Une valeur de code de réponse zéro n'est pas considérée comme un code de réponse d'erreur. Chaque code de réponse et type de code de réponse est défini dans l'EDD.

Structure lexicale

```
fail_on_response_code(code)
```

Les éléments lexicaux du Builtin fail_on_response_code sont présentés dans le Tableau 97.

Tableau 97 – Builtin fail_on_response_code

Nom du paramètre	Type	Direction	Usage	Description
code	unsigned long	I	m	Code de réponse.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.95 Builtin fassign

Objet

Le Builtin fassign assigne une valeur à une VARIABLE.

Structure lexicale

```
fassign(device_var, new_value)
```

Les éléments lexicaux du Builtin fassign sont présentés dans le Tableau 98.

Tableau 98 – Builtin fassign

Nom du paramètre	Type	Direction	Usage	Description
device_var	reference	I	m	Nom de la VARIABLE.
new_value	double	I	m	Nouvelle valeur de la VARIABLE référencée.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau 367.

4.96 Builtin fGetByte

Objet

Le Builtin fGetByte permet d'accéder au contenu BLOB à des fins de validation uniquement.

Structure lexicale

```
fGetByte(blob_item, position, value_read)
```

Les éléments lexicaux du Builtin fGetByte sont présentés dans le Tableau 99.

Tableau 99 – Builtin fGetByte

Nom du paramètre	Type	Direction	Usage	Description
blob_item	DD_ITEM &	I	m	Contenu BLOB duquel l'octet doit être extrait.
position	unsigned long long	I	m	Position par rapport au début de la lecture.
value_read	unsigned char &	O	m	Valeur issue du contenu BLOB.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau 367.

4.97 Builtin fgetval

Objet

Le Builtin fgetval est spécifiquement conçu pour être utilisé dans des méthodes prélecture/écriture et postlecture/écriture. Cette fonction retourne la valeur d'une VARIABLE du dispositif comme si elle avait été reçue depuis le dispositif connecté. Des opérations de mise à l'échelle peuvent ensuite être effectuées sur la valeur de VARIABLE avant qu'elle ne soit stockée dans l'application EDD.

Structure lexicale

fgetval (VOID)

L'élément lexical du Builtin fgetval est présenté dans le Tableau 100.

Tableau 100 – Builtin fgetval

Nom du paramètre	Type	Direction	Usage	Description
<return>	double	O	m	Valeur de la variable.

4.98 Builtin float_value

Objet

Le Builtin float_value retourne la valeur de la VARIABLE spécifiée.

Structure lexicale

float_value(source_var_name)

Les éléments lexicaux du Builtin float_value sont présentés dans le Tableau 101.

Tableau 101 – Builtin float_value

Nom du paramètre	Type	Direction	Usage	Description
source_var_name	reference	I	m	Nom de la référence de VARIABLE.
<return>	double	O	m	Valeur de la variable.

4.99 Builtin FloatToByte

Objet

Le Builtin FloatToByte fournit un vecteur de valeurs de caractère non signé représentant les éléments particuliers d'une valeur à virgule flottante.

Structure lexicale

```
FloatToByte(in, out1, out2, out3, out4)
```

Les éléments lexicaux du Builtin FloatToByte sont présentés dans le Tableau 102.

Tableau 102 – Builtin FloatToByte

Nom du paramètre	Type	Direction	Usage	Description
in	float	I	m	Valeur à virgule flottante à convertir.
out1	unsigned char	O	m	Premier élément de la valeur à virgule flottante (octet de poids fort).
out2	unsigned char	O	m	Deuxième élément de la valeur à virgule flottante.
out3	unsigned char	O	m	Troisième élément de la valeur à virgule flottante.
out4	unsigned char	O	m	Quatrième élément de la valeur à virgule flottante (octet de poids faible).
<return>	VOID	—	—	—

4.100 Builtin floor**Objet**

Le Builtin floor (valeur plancher) retourne la valeur entière la plus élevée inférieure ou égale à une valeur à virgule flottante spécifiée.

Structure lexicale

```
floor(x)
```

Les éléments lexicaux du Builtin floor sont présentés dans le Tableau 103.

Tableau 103 – Builtin floor

Nom du paramètre	Type	Direction	Usage	Description
x	double	I	m	Valeur à virgule flottante.
<return>	double	O	m	Valeur plancher.

4.101 Builtin fmod**Objet**

Le Builtin fmod retourne le résidu de virgule flottante de la division de deux valeurs à virgule flottante.

Structure lexicale

```
fmod(x, y)
```

Les éléments lexicaux du Builtin fmod sont présentés dans le Tableau 104.

Tableau 104 – Builtin fmod

Nom du paramètre	Type	Direction	Usage	Description
x	double	I	m	Numérateur.
y	double	I	m	Dénominateur.
<return>	double	O	m	Reste.

4.102 Builtin fpclassify

Objet

Le Builtin fpclassify classe une valeur à virgule flottante comme n'étant pas un nombre, l'infini, zéro, une valeur sous-normale ou normale.

NOTE Ce Builtin est implémenté comme une macro qui génère un appel de fpclassifyf ou fpclassifyd suivant que son paramètre d'entrée est une valeur à virgule flottante ou une valeur à double précision, respectivement.

Structure lexicale

fpclassify(value)

Les éléments lexicaux du Builtin fpclassify sont présentés dans le Tableau 105.

Tableau 105 – Builtin fpclassifyd

Nom du paramètre	Type	Direction	Usage	Description
value	Float ou double	I	m	Valeur à virgule flottante à classer.
<return>	int	O	m	FP_NAN lorsque la valeur du paramètre n'est pas un nombre (NAN = "Not a Number"). FP_INFINITE lorsque la valeur du paramètre est plus ou moins l'infini. FP_ZERO lorsque la valeur du paramètre est zéro positif ou négatif. FP_SUBNORMAL lorsque la valeur du paramètre est dénormalisée. Les nombres dénormalisés ont des valeurs qui sont trop faibles (c'est-à-dire, des valeurs très proches de zéro) pour être représentées autrement. FP_NORMAL est retourné pour toutes les autres valeurs du paramètre

4.103 Builtin From_DATE_AND_TIME_VALUE

Objet

Le Builtin From_DATE_AND_TIME_VALUE crée une valeur time_t à partir d'une DATE et une TIME_VALUE(4).

Structure lexicale

From_DATE_AND_TIME_VALUE(date, time_value)

Les éléments lexicaux du Builtin From_DATE_AND_TIME_VALUE sont présentés dans le Tableau 106.

Tableau 106 – Builtin From_DATE_AND_TIME_VALUE

Nom du paramètre	Type	Direction	Usage	Description
date	long	I	m	Valeur de DATE.
time_value	unsigned long	I	m	Valeur de TIME_VALUE(4).
<return>	time_t	O	m	Valeur time_t créée ou -1 si un paramètre non valide est détecté.

4.104 Builtin From_TIME_VALUE**Objet**

Le Builtin From_TIME_VALUE crée une valeur time_t à partir d'une TIME_VALUE(4) et de la date actuelle (à minuit) ou d'une TIME_VALUE(8).

Structure lexicale

From_TIME_VALUE(time_value)

Les éléments lexicaux du Builtin From_TIME_VALUE sont présentés dans le Tableau 107.

Tableau 107 – Builtin From_TIME_VALUE

Nom du paramètre	Type	Direction	Usage	Description
time_value	unsigned long ou unsigned long long	I	m	Valeur de TIME_VALUE. Une TIME_VALUE(8) est passée comme un unsigned long long et est un temps absolu. Une TIME_VALUE(4) est passée comme un unsigned long et est une heure de la journée
<return>	time_t	O	m	Valeur time_t créée ou -1 si un paramètre non valide est détecté.

4.105 Builtin fsetval**Objet**

Le Builtin fsetval définit la valeur d'une VARIABLE de type virgule flottante pour laquelle une action pré/post a été désignée à la valeur spécifiée.

La fonction fsetval est spécifiquement conçue pour être utilisée dans des méthodes prélecture/écriture et postlecture/écriture. Cette fonction définit la valeur d'une VARIABLE du dispositif à la valeur spécifiée.

Structure lexicale

fsetval(value)

Les éléments lexicaux du Builtin fsetval sont présentés dans le Tableau 108.

Tableau 108 – Builtin fsetval

Nom du paramètre	Type	Direction	Usage	Description
value	double	I	m	Nouvelle valeur pour la VARIABLE.
<return>	double	O	m	Valeur de la variable.

4.106 Builtin ftoa

Objet

Le Builtin convertit une valeur à virgule flottante en chaîne de texte.

Structure lexicale

`ftoa(f)`

Les éléments lexicaux du Builtin ftoa sont présentés dans le Tableau 109.

Tableau 109 – Builtin ftoa

Nom du paramètre	Type	Direction	Usage	Description
<code>f</code>	float	I	m	Valeur à virgule flottante à convertir en texte.
<code><return></code>	char[]	O	m	Valeur à virgule flottante sous forme de chaîne.

4.107 Builtin fvar_value

Objet

Le Builtin `fvar_value` retourne la valeur de la VARIABLE spécifiée.

Structure lexicale

`fvar_value(source_var)`

Les éléments lexicaux du Builtin `fvar_value` sont présentés dans le Tableau 110.

Tableau 110 – Builtin fvar_value

Nom du paramètre	Type	Direction	Usage	Description
<code>source_var</code>	reference	I	m	Nom de la référence de VARIABLE.
<code><return></code>	double	O	m	Valeur de la variable.

4.108 Builtin get_acknowledgement

Objet

Le Builtin `get_acknowledgement` affiche le message d'invite fourni sur le dispositif d'affichage de l'application et retourne à la méthode après que l'utilisateur ait accusé réception du message. Les valeurs de variable peuvent être incorporées dans la chaîne de message avec des informations de formatage.

Structure lexicale

`get_acknowledgement(prompt, ids, indices, id_count)`

Les éléments lexicaux du Builtin `get_acknowledgement` sont présentés dans le Tableau 111.

Tableau 111 – Builtin get_acknowledgement

Nom du paramètre	Type	Direction	Usage	Description
prompt	char[]	I	m	Message à afficher.
ids	array of unsigned long	I	m	ID d'élément des VARIABLES utilisées dans le message d'invite.
indices	array of unsigned long	I	m	ID de membre ou index de matrice des variables utilisées dans le message d'invite. Défini à zéro s'il n'y a pas d>ID de membre ou d'index de matrice.
id_count	long	I	m	Nombre d'éléments de matrice dans les matrices pour les ID et les index. Défini à zéro si les matrices pour les ID et les index sont NULL.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.109 Builtin get_acknowledgement2**Objet**

Le Builtin get_acknowledgement2 affiche le message d'invite fourni sur le dispositif d'affichage de l'application et retourne à la méthode après que l'utilisateur ait accusé réception du message. Les valeurs de variable peuvent être incorporées dans la chaîne de message avec des informations de formatage.

Structure lexicale

```
get_acknowledgement2(prompt, blk_ids, blk_nums, ids, indices, id_count)
```

Les éléments lexicaux du Builtin get_acknowledgement2 sont présentés dans le Tableau 112.

Tableau 112 – Builtin get_acknowledgement2

Nom du paramètre	Type	Direction	Usage	Description
prompt	char[]	I	m	Message à afficher.
blk_ids	array of unsigned long	I	m	ID d'élément des instances de BLOCK_A utilisées dans le message d'invite.
blk_nums	array of unsigned long	I	m	Numéros d'occurrence des instances de BLOCK_A utilisées dans le message d'invite.
ids	array of unsigned long	I	m	ID d'élément des VARIABLES utilisées dans le message d'invite.
indices	array of unsigned long	I	m	ID de membre ou index de matrice des variables utilisées dans le message d'invite. Défini à zéro s'il n'y a pas d>ID de membre ou d'index de matrice.
id_count	long	I	m	Nombre d'éléments de matrice dans les matrices pour les ID et les index. Défini à zéro si les matrices pour les ID et les index sont NULL.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.110 Builtin get_block_instance_by_objet_index**Objet**

Le Builtin get_block_instance_by_objet_index retourne le numéro d'occurrence de l'instance de BLOCK_A à l'index d'objet spécifié.

Structure lexicale

`get_block_instance_by_object_index(obj_idx, blk_num)`

Les éléments lexicaux du Builtin `get_block_instance_by_object_index` sont présentés dans le Tableau 113.

Tableau 113 – Builtin `get_block_instance_by_object_index`

Nom du paramètre	Type	Direction	Usage	Description
obj_idx	unsigned long	I	m	Index d'objet de l'instance de BLOCK_A à retourner.
blk_num	unsigned long	O	m	Numéro d'occurrence de l'instance de BLOCK_A spécifiée.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.111 Builtin `get_block_instance_by_tag`

Objet

Le Builtin `get_block_instance_by_tag` retourne le numéro d'occurrence de l'instance de BLOCK_A avec la balise spécifiée. Si le paramètre `blockId` est égal à zéro, le numéro d'occurrence de bloc dont le Block Tag (balise de bloc) correspond au paramètre `blockTag` est retourné. Dans le cas contraire, le numéro d'occurrence du bloc dont l'Identifiant d'élément DD correspond au paramètre `blockId` et dont le Block Tag correspond au paramètre `blockTag` est retourné. Lorsque plusieurs blocs satisfont aux critères spécifiés, le numéro d'occurrence du bloc avec le plus petit index d'objet est retourné.

Structure lexicale

`get_block_instance_by_tag(blockId, blockTag, blk_num)`

Les éléments lexicaux du Builtin `get_block_instance_by_tag` sont présentés dans le Tableau 114.

Tableau 114 – Builtin `get_block_instance_by_tag`

Nom du paramètre	Type	Direction	Usage	Description
blockId	unsigned long	I	m	ID d'élément de l'instance de BLOCK_A à retourner.
blockTag	char[]	I	m	Balise de l'instance de BLOCK_A à retourner.
blk_num	unsigned long	O	m	Numéro d'occurrence de l'instance de BLOCK_A spécifiée.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.112 Builtin `get_block_instance_count`

Objet

Le Builtin `get_block_instance_count` retourne le nombre d'instances de BLOCK_A qui sont instanciées dans le dispositif. Si le paramètre `blockId` est égal à zéro, le nombre total de blocs actuellement instanciés dans le dispositif est retourné. Dans le cas contraire, le nombre de blocs dont l'ID d'élément DD correspond au paramètre `blockId` est retourné.

Structure lexicale

`get_block_instance_count(blockId, count)`

Les éléments lexicaux du Builtin `get_block_instance_count` sont présentés dans le Tableau 115.

Tableau 115 – Builtin `get_block_instance_count`

Nom du paramètre	Type	Direction	Usage	Description
<code>blockId</code>	unsigned long	I	m	ID d'élément des instances de BLOCK_A à compter.
<code>count</code>	unsigned long	O	m	Nombre d'instances de BLOCK_A du type spécifié qui sont instanciées dans le dispositif.
<code><return></code>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.113 Builtin `get_comm_error`

Objet

Le Builtin `get_comm_error` retourne le code d'erreur de communication reçu depuis une application en cas de code de retour de Builtin `BLTIN_FAIL_COMM_ERROR`.

Ce Builtin est principalement utilisé à des fins de débogage pour déterminer le type d'erreurs de communication qui se produisent en affichant l'erreur de communication afin que l'auteur de la méthode puisse la visualiser.

Dans de rares cas, la méthode peut réagir à des erreurs de communication spécifiques. Cependant, il convient que la méthode spécifie une gestion alternative des erreurs de communication comme suit:

- `fail_on_all_comm_errors`
- `fail_on_comm_error`
- `abort_on_all_comm_errors`
- `ABORT_ON_COMM_ERROR`

La gestion alternative peut conduire à l'interruption de la méthode ou au traitement immédiat de l'erreur sur la base de la présence d'une erreur de communication.

Les codes d'erreur de communication sont définis par les consortiums. Leurs chaînes associées peuvent être définies dans un dictionnaire de texte. Une erreur de communication zéro n'est pas considérée comme une erreur de communication. Le Builtin obtient le code d'erreur de communication retourné par la dernière demande de communication.

Structure lexicale

`get_comm_error(VOID)`

L'élément lexical du Builtin `get_comm_error` est présenté dans le Tableau 116.

Tableau 116 – Builtin `get_comm_error`

Nom du paramètre	Type	Direction	Usage	Description
<code><return></code>	unsigned long	O	m	Code d'erreur de communication.

4.114 Builtin get_comm_error_string

Objet

Le Builtin get_comm_error_string formate une chaîne descriptive décrivant l'erreur de communication sur la base du code d'erreur qui est fourni. Si possible, la description de l'erreur de communication est utilisée; si la description n'est pas disponible, un message d'erreur de communication par défaut avec le numéro d'erreur est utilisé.

Structure lexicale

```
get_comm_error_string(error, string, maxlen)
```

Les éléments lexicaux du Builtin get_comm_error_string sont présentés dans le Tableau 117.

Tableau 117 – Builtin get_comm_error_string

Nom du paramètre	Type	Direction	Usage	Description
error	unsigned long	I	m	Code d'erreur de communication.
string	char[]	I	m	Description de l'erreur de communication.
maxlen	long	I	m	Longueur maximale de la chaîne.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.115 Builtin get_date

Objet

Le Builtin get_date extrait la valeur actuelle de date non mise à l'échelle de la VARIABLE traitée. Le Builtin est appelé uniquement par les méthodes associées à une action prédiction, postédition, postlecture, ou préécriture.

Les méthodes associées aux actions prédiction, postédition, postlecture ou préécriture sont exécutées lorsqu'une VARIABLE pour laquelle une de ces actions est définie en tant qu'attribut est lue ou écrite par une application, une méthode d'utilisateur ou une méthode d'édition.

Structure lexicale

```
get_date(data, size)
```

Les éléments lexicaux du Builtin get_date sont présentés dans le Tableau 118.

Tableau 118 – Builtin get_date

Nom du paramètre	Type	Direction	Usage	Description
données(data)	char[]	O	m	Valeur retournée de la VARIABLE. La structure de cette mémoire tampon d'octet est définie dans cette spécification. Ce paramètre de sortie doit pointer vers un stockage valide.
size	long	O	m	Longueur retournée de la VARIABLE. Ce paramètre de sortie doit pointer vers un stockage valide.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.116 Builtin get_date_lelem

Objet

Le Builtin get_date_lelem extrait une valeur de date d'une LIST.

Structure lexicale

```
get_date_lelem(list_id, index, element_id, subelement_id, data, size)
```

Les éléments lexicaux du Builtin get_date_lelem sont présentés dans le Tableau 119.

Tableau 119 – Builtin get_date_lelem

Nom du paramètre	Type	Direction	Usage	Description
list_id	unsigned long	I	m	ID d'élément de la liste.
index	int	I	m	L'index de la liste contenant l'élément à extraire.
element_id	unsigned long	I	m	ID d'élément de l'élément à extraire.
subelement_id	unsigned long	I	m	ID de membre de l'élément à extraire.
data	char *	O	m	Mémoire tampon dans laquelle les données sont à retourner.
size	int *	O	m	Taille des données retournées.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.117 Builtin get_date_lelem2

Objet

Le Builtin get_date_lelem2 extrait une valeur de date d'une liste incorporée.

Structure lexicale

```
get_date_lelem2(list_id, index, embedded_list_id, embedded_list_index,
element_id, subelement_id, data, size)
```

Les éléments lexicaux du Builtin get_date_lelem2 sont présentés dans le Tableau 120.

Tableau 120 – Builtin get_date_lelem2

Nom du paramètre	Type	Direction	Usage	Description
list_id	unsigned long	I	m	ID d'élément de la liste.
index	int	I	m	L'index de la liste contenant l'élément à extraire.
embedded_list_id	unsigned long	I	m	ID d'élément de la liste incorporée.
embedded_list_index	int	I	m	Index de la liste incorporée contenant l'élément à extraire.
element_id	unsigned long	I	m	ID d'élément de l'élément à extraire.
subelement_id	unsigned long	I	m	ID de membre de l'élément à extraire.
data	char *	O	m	Mémoire tampon dans laquelle les données sont à retourner.
size	int *	O	m	Taille des données retournées.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.118 Builtin get_date_value**Objet**

Le Builtin get_date_value extrait la valeur actuelle mise à l'échelle de la VARIABLE spécifiée de la mémoire cache ou du dispositif et retourne celle-ci.

La valeur est une mémoire tampon d'octet qui peut contenir jusqu'à huit caractères et une VARIABLE de type date-et-heure, heure ou durée du dispositif.

Structure lexicale

```
get_date_value(id, member_id, data, size)
```

Les éléments lexicaux du Builtin get_date_value sont présentés dans le Tableau 121.

Tableau 121 – Builtin get_date_value

Nom du paramètre	Type	Direction	Usage	Description
ID	unsigned long	I	m	ID d'une référence de VARIABLE.
member_id	unsigned long	I	m	ID de membre d'un BLOCK_A, PARAMETER_LIST, RECORD, VALUE_ARRAY, COLLECTION ou REFERENCE_ARRAY.
data	char[]	O	m	Mémoire tampon de 8 octets max. pour le résultat du Builtin.
size	long	O	m	Taille de la mémoire tampon utilisée.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.119 Builtin get_date_value2**Objet**

Le Builtin get_date_value2 extrait la valeur actuelle mise à l'échelle de la VARIABLE spécifiée de la mémoire cache ou du dispositif et retourne celle-ci.

La valeur est une mémoire tampon d'octets qui peut contenir jusqu'à huit caractères et une VARIABLE de type date-et-heure, heure ou durée du dispositif.

Structure lexicale

```
get_date_value2(blk_id, blk_num, id, member_id, data, size)
```

Les éléments lexicaux du Builtin get_date_value2 sont présentés dans le Tableau 122.

Tableau 122 – Builtin get_date_value2

Nom du paramètre	Type	Direction	Usage	Description
blk_id	unsigned long	I	m	ID d'élément de l'instance de BLOCK_A à accéder.
blk_num	unsigned long	I	m	Numéro d'occurrence de l'instance de BLOCK_A à accéder.
id	unsigned long	I	m	ID d'une référence de VARIABLE.
member_id	unsigned long	I	m	ID de membre d'un BLOCK_A, PARAMETER_LIST, RECORD, VALUE_ARRAY, COLLECTION ou REFERENCE_ARRAY.
data	char[]	O	m	Mémoire tampon de 8 octets max. pour le résultat du Builtin.
size	long	O	m	Taille de la mémoire tampon utilisée.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.120 Builtin GET_DD_REVISION

Objet

Le Builtin GET_DD_REVISION extrait la DD_REVISION à partir de la description du dispositif actuellement utilisé comme indiqué dans l'identification de la description du dispositif avec le mot clé DD_REVISION.

Structure lexicale

```
GET_DD_REVISION (VOID)
```

L'élément lexical du Builtin GET_DD_REVISION est présenté dans le Tableau 123.

Tableau 123 – Builtin GET_DD_REVISION

Nom du paramètre	Type	Direction	Usage	Description
<return>	int	o	m	DD_REVISION actuelle.

4.121 Builtin get_dds_error

Objet

Le Builtin get_dds_error constitue un moyen de débogage des appels de sous-programmes Builtin qui retournent une erreur. Il convient de ne jamais l'utiliser dans une méthode de production finale, parce qu'il convient que les types des erreurs retournées avec get_dds_error ne soient jamais reçus dans une méthode totalement déboguée.

Une méthode peut déterminer l'erreur actuelle de l'EDD après la réception du retour d'erreur BLTIN_DDS_ERROR est reçu en appelant ce Builtin. Une représentation textuelle du message d'erreur est intégrée dans une mémoire tampon fournie avec une longueur maximale donnée.

L'erreur de l'EDD et la description facultative retournées concernant l'erreur de l'EDD la plus récente.

Retourne l'erreur de l'EDD générée par un appel précédent et retourne éventuellement une chaîne de texte décrivant l'erreur.

Structure lexicale

`get_dds_error(string, maxlen)`

Les éléments lexicaux du Builtin `get_dds_error` sont présentés dans le Tableau 124.

Tableau 124 – Builtin `get_dds_error`

Nom du paramètre	Type	Direction	Usage	Description
string	char[]	O	m	Mémoire tampon de sortie pour la description de l'erreur.
maxlen	long	I	m	Longueur maximale de la mémoire tampon de sortie (chaîne) en octets.
<return>	long	O	m	Code d'erreur généré par un appel de Builtin précédent.

4.122 Builtin `GET_DEV_VAR_VALUE`

Objet

Le Builtin `GET_DEV_VAR_VALUE` affiche le message d'invite spécifié et permet à l'utilisateur d'éditer la valeur d'une VARIABLE du dispositif. La copie éditée de la valeur de VARIABLE du dispositif est mise à jour lorsque la nouvelle valeur est entrée, mais n'est pas envoyée au dispositif. Cela doit être effectué de manière explicite en utilisant une des fonctions de commande d'envoi.

L'invite peut NE PAS contenir des valeurs incorporées de VARIABLE locales et/ou du dispositif.

NOTE 1 Il n'est pas garanti que les messages précédemment affichés restent à l'écran.

NOTE 2 Certaines applications EDD effacent l'affichage à chaque appel de cette fonction.

Structure lexicale

`GET_DEV_VAR_VALUE(prompt, device_var_name)`

Les éléments lexicaux du Builtin `GET_DEV_VAR_VALUE` sont présentés dans le Tableau 125.

Tableau 125 – Builtin `GET_DEV_VAR_VALUE`

Nom du paramètre	Type	Direction	Usage	Description
prompt	char[]	I	m	Message à afficher.
device_var_name	reference	I	m	Nom référencé de la VARIABLE, qui peut être éditée.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau 367.

4.123 Builtin get_dev_var_value

Objet

Le Builtin get_dev_var_value affiche le message d'invite spécifié et permet à l'utilisateur d'éditer la valeur d'une VARIABLE du dispositif. La copie éditée de la valeur de VARIABLE du dispositif est mise à jour lorsque la nouvelle valeur est entrée mais n'est pas envoyée au dispositif. Cela doit être effectué de manière explicite en utilisant une des fonctions de commande d'envoi.

L'invite peut contenir des valeurs de VARIABLE locales et/ou du dispositif incorporées (voir le Builtin PUT_MESSAGE).

NOTE 1 Il n'est pas garanti que les messages précédemment affichés restent à l'écran.

NOTE 2 Certaines applications EDD effacent l'affichage à chaque appel de cette fonction.

Structure lexicale

get_dev_var_value(prompt, global_var_ids, device_var_name)

Les éléments lexicaux du Builtin get_dev_var_value sont présentés dans le Tableau 126.

Tableau 126 – Builtin get_dev_var_value

Nom du paramètre	Type	Direction	Usage	Description
prompt	char[]	I	m	Message à afficher.
global_var_ids	array of int	I	m	Matrice qui contient des identifiants uniques d'instance de VARIABLE.
device_var_name	reference	I	m	Nom de la VARIABLE, qui peut être édité.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau 367.

4.124 Builtin GET_DEVICE_REVISION

Objet

Le Builtin GET_DEVICE_REVISION extrait la DEVICE_REVISION (version du dispositif) à partir de la description du dispositif actuellement utilisée comme indiqué dans l'identification de la description du dispositif avec le mot clé DEVICE_REVISION.

Structure lexicale

GET_DEVICE_REVISION (VOID)

L'élément lexical du Builtin GET_DEVICE_REVISION est présenté dans le Tableau 127.

Tableau 127 – Builtin GET_DEVICE_REVISION

Nom du paramètre	Type	Direction	Usage	Description
<return>	int	o	m	DEVICE_REVISION actuelle.

4.125 Builtin GET_DEVICE_TYPE

Objet

Le Builtin GET_DEVICE_TYPE extrait l'équivalent numérique du DEVICE_TYPE (type de dispositif) de la description du dispositif actuellement utilisée comme indiqué dans l'identification de la description du dispositif avec le mot clé DEVICE_TYPE.

Structure lexicale

GET_DEVICE_TYPE (VOID)

L'élément lexical du Builtin GET_DEVICE_TYPE est présenté dans le Tableau 128.

Tableau 128 – Builtin GET_DEVICE_TYPE

Nom du paramètre	Type	Direction	Usage	Description
<return>	int	o	m	DEVICE_TYPE actuel.

4.126 Builtin get_dictionary_string

Objet

Le Builtin get_dictionary_string extrait la chaîne de dictionnaire de texte associée au nom donné dans la langue actuelle. Si la chaîne n'est pas disponible dans la langue actuelle, la chaîne en anglais est extraite. Si la chaîne n'est définie dans aucune langue, cela crée une condition d'erreur et la fonction retourne FALSE. Si la chaîne est plus longue que la longueur maximale max_str_len, la chaîne est tronquée. Le paramètre dict_string_name est le nom de la chaîne tel qu'il apparaît dans le dictionnaire de texte.

Structure lexicale

get_dictionary_string(dict_string_name, string, max_str_len)

Les éléments lexicaux du Builtin get_dictionary_string sont présentés dans le Tableau 129.

Tableau 129 – Builtin get_dictionary_string

Nom du paramètre	Type	Direction	Usage	Description
dict_string_name	reference	l	m	Nom d'une entrée de dictionnaire.
string	char[]	l	m	Mémoire tampon pour la chaîne de résultat.
max_str_len	int	o	m	Longueur maximale de la mémoire tampon.
<return>	int	o	m	Un des codes de retour spécifiés dans le Tableau 367.

4.127 Builtin get_double

Objet

Le Builtin get_double extrait la valeur double précision actuelle non mise à l'échelle de la VARIABLE traitée.

Il obtient la valeur actuelle de la VARIABLE double précision comme si elle était lue directement depuis un dispositif.

Le Builtin get_double est appelé uniquement par des méthodes associées à une action préédition, postédition, postlecture ou préécriture.

Les méthodes associées aux actions préédition, postédition, postlecture ou préécriture sont exécutées lorsqu'une VARIABLE pour laquelle une de ces actions est définie en tant qu'attribut est lue par une application ou une autre méthode.

Structure lexicale

get_double(value)

Les éléments lexicaux du Builtin get_double sont présentés dans le Tableau 130.

Tableau 130 – Builtin get_double

Nom du paramètre	Type	Direction	Usage	Description
value	double	O	m	Valeur retournée de la VARIABLE.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.128 Builtin get_double_lelem**Objet**

Le Builtin get_double_lelem extrait une valeur double précision à partir d'une LIST.

Structure lexicale

```
get_double_lelem(list_id, index, element_id, subelement_id)
```

Les éléments lexicaux du Builtin get_double_lelem sont présentés dans le Tableau 131.

Tableau 131 – Builtin get_double_lelem

Nom du paramètre	Type	Direction	Usage	Description
list_id	unsigned long	I	m	ID d'élément de la liste.
index	int	I	m	L'index de la liste contenant l'élément à extraire.
element_id	unsigned long	I	m	ID d'élément de l'élément à extraire.
subelement_id	unsigned long	I	m	ID de membre de l'élément à extraire.
<return>	double	O	m	Valeur spécifiée.

4.129 Builtin get_double_lelem2**Objet**

Le Builtin get_double_lelem2 extrait une valeur double précision à partir d'une liste incorporée.

Structure lexicale

```
get_double_lelem2(list_id, index, embedded_list_id, embedded_list_index, element_id, subelement_id)
```

Les éléments lexicaux du Builtin get_double_lelem2 sont présentés dans le Tableau 132.

Tableau 132 – Builtin get_double_lelem2

Nom du paramètre	Type	Direction	Usage	Description
list_id	unsigned long	I	m	ID d'élément de la liste.
index	int	I	m	L'index de la liste contenant l'élément à extraire.
embedded_list_id	unsigned long	I	m	ID d'élément de la liste incorporée.
embedded_list_index	int	I	m	Index de la liste incorporée contenant l'élément à extraire.
element_id	unsigned long	I	m	ID d'élément de l'élément à extraire.
subelement_id	unsigned long	I	m	ID de membre de l'élément à extraire.
<return>	double	O	m	Valeur spécifiée.

4.130 Builtin get_double_value

Objet

Le Builtin get_double_value extrait la valeur actuelle mise à l'échelle de la VARIABLE spécifiée de la mémoire cache ou du dispositif et retourne celle-ci.

Les id de paramètre et member_id sont spécifiés en utilisant les Builtin ITEM_ID et MEMBER_ID et le nom ou la référence de la variable.

La VARIABLE doit avoir un type de données double précision et être une variable valide dans la mémoire cache ou le dispositif.

Structure lexicale

```
get_double_value(id, member_id, value)
```

Les éléments lexicaux du Builtin get_double_value sont présentés dans le Tableau 133.

Tableau 133 – Builtin get_double_value

Nom du paramètre	Type	Direction	Usage	Description
id	unsigned long	I	m	ID d'élément de la VARIABLE à accéder.
member_id	unsigned long	I	m	ID de membre de la VARIABLE à accéder.
value	double	O	m	Valeur retournée de la VARIABLE.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.131 Builtin get_double_value2

Objet

Le Builtin get_double_value2 extrait la valeur actuelle mise à l'échelle de la VARIABLE spécifiée de la mémoire cache ou du dispositif et retourne celle-ci.

Les id de paramètre et member_id sont spécifiés en utilisant les Builtin ITEM_ID et MEMBER_ID et le nom ou la référence de la variable.

La VARIABLE doit avoir un type de données double précision et être une variable valide dans la mémoire cache ou le dispositif.

Structure lexicale

```
get_double_value2(blk_id, blk_num, id, member_id, value)
```

Les éléments lexicaux du Builtin get_double_value2 sont présentés dans le Tableau 134.

Tableau 134 – Builtin get_double_value2

Nom du paramètre	Type	Direction	Usage	Description
blk_id	unsigned long	I	m	ID d'élément de l'instance de BLOCK_A à accéder.
blk_num	unsigned long	I	m	Numéro d'occurrence de l'instance de BLOCK_A à accéder.
id	unsigned long	I	m	ID d'élément de la VARIABLE à accéder.
member_id	unsigned long	I	m	ID de membre de la VARIABLE à accéder.
value	double	O	m	Valeur retournée de la VARIABLE.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.132 Builtin get_enum_string

Objet

Le Builtin get_enum_string retourne la chaîne associée à la valeur d'une variable énumérée ou binaire énumérée.

Structure lexicale

```
get_enum_string(variable, value, string)
```

Les éléments lexicaux du Builtin get_enum_string sont présentés dans le Tableau 135.

Tableau 135 – Builtin get_enum_string

Nom du paramètre	Type	Direction	Usage	Description
variable	DD_ITEM	I	m	Variable énumérée ou binaire énumérée.
value	int	I	m	Valeur de la variable.
string	DD_STRING	O	m	Chaîne associée à la valeur de la variable.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.133 Builtin get_float

Objet

Le Builtin get_float extrait la valeur à virgule flottante actuelle non mise à l'échelle de la VARIABLE traitée.

Il extrait la valeur actuelle d'une VARIABLE à virgule flottante non mise à l'échelle comme si elle était lue directement depuis un dispositif.

Le Builtin `get_float` peut être appelé uniquement par des méthodes associées à une action prédédition, postédition, postlecture ou préécriture. Les méthodes associées aux actions prédédition, postédition, postlecture ou préécriture sont exécutées lorsqu'une VARIABLE pour laquelle une de ces actions est définie en tant qu'attribut est lue par une application, une méthode utilisateur ou une méthode d'édition.

Structure lexicale

`get_float(value)`

Les éléments lexicaux du Builtin `get_float` sont présentés dans le Tableau 136.

Tableau 136 – Builtin `get_float`

Nom du paramètre	Type	Direction	Usage	Description
value	float	O	m	Valeur retournée de la VARIABLE.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.134 Builtin `get_float_lelem`

Objet

Le Builtin `get_float_lelem` extrait une valeur à virgule flottante à partir d'une LIST.

Structure lexicale

`get_float_lelem(list_id, index, element_id, subelement_id)`

Les éléments lexicaux du Builtin `get_float_lelem` sont présentés dans le Tableau 137.

Tableau 137 – Builtin `get_float_lelem`

Nom du paramètre	Type	Direction	Usage	Description
list_id	unsigned long	I	m	ID d'élément de la liste.
index	int	I	m	L'index de la liste contenant l'élément à extraire.
element_id	unsigned long	I	m	ID d'élément de l'élément à extraire.
subelement_id	unsigned long	I	m	ID de membre de l'élément à extraire.
<return>	float	O	m	Valeur spécifiée.

4.135 Builtin `get_float_lelem2`

Objet

Le Builtin `get_float_lelem2` extrait une valeur à virgule flottante à partir d'une liste incorporée.

Structure lexicale

`get_float_lelem2(list_id, index, embedded_list_id, embedded_list_index, element_id, subelement_id)`

Les éléments lexicaux du Builtin `get_float_lelem2` sont présentés dans le Tableau 138.

Tableau 138 – Builtin get_float_lelem2

Nom du paramètre	Type	Direction	Usage	Description
list_id	unsigned long	I	m	ID d'élément de la liste.
index	int	I	m	L'index de la liste contenant l'élément à extraire.
embedded_list_id	unsigned long	I	m	ID d'élément de la liste incorporée.
embedded_list_index	int	I	m	Index de la liste incorporée contenant l'élément à extraire.
element_id	unsigned long	I	m	ID d'élément de l'élément à extraire.
subelement_id	unsigned long	I	m	ID de membre de l'élément à extraire.
<return>	float	O	m	Valeur spécifiée.

4.136 Builtin get_float_value**Objet**

Le Builtin `get_float_value` extrait la valeur actuelle mise à l'échelle de la VARIABLE spécifiée de la mémoire cache ou du dispositif et retourne celle-ci.

L'id et le member_id sont spécifiés en utilisant les Builtin `ITEM_ID` et `MEMBER_ID` et le nom ou la référence de la variable.

Structure lexicale

```
get_float_value(id, member_id, value)
```

Les éléments lexicaux du Builtin `get_float_value` sont présentés dans le Tableau 139.

Tableau 139 – Builtin get_float_value

Nom du paramètre	Type	Direction	Usage	Description
id	unsigned long	I	m	ID d'élément de la VARIABLE à accéder.
member_id	unsigned long	I	m	ID de membre de la VARIABLE à accéder.
value	float	O	m	Valeur retournée de la VARIABLE.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.137 Builtin get_float_value2**Objet**

Le Builtin `get_float_value2` extrait la valeur actuelle mise à l'échelle de la VARIABLE spécifiée de la mémoire cache ou du dispositif et retourne celle-ci.

L'id et le member_id sont spécifiés en utilisant les Builtin `ITEM_ID` et `MEMBER_ID` et le nom ou la référence de la variable.

Structure lexicale

```
get_float_value2(id, member_id, value)
```

Les éléments lexicaux du Builtin `get_float_value2` sont présentés dans le Tableau 140.

Tableau 140 – Builtin get_float_value2

Nom du paramètre	Type	Direction	Usage	Description
blk_id	unsigned long	I	m	ID d'élément de l'instance de BLOCK_A à accéder.
blk_num	unsigned long	I	m	Numéro d'occurrence de l'instance de BLOCK_A à accéder.
id	unsigned long	I	m	ID d'élément de la VARIABLE à accéder.
member_id	unsigned long	I	m	ID de membre de la VARIABLE à accéder.
value	float	O	m	Valeur retournée de la VARIABLE.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.138 Builtin GET_LOCAL_VAR_VALUE

Objet

Le Builtin GET_LOCAL_VAR_VALUE affiche le message d'invite spécifié et permet à l'utilisateur d'éditer la valeur d'une VARIABLE locale.

L'invite peut NE PAS contenir des valeurs incorporées de VARIABLE locales et/ou du dispositif.

NOTE 1 Il n'est pas garanti que les messages précédemment affichés restent à l'écran.

NOTE 2 Certaines applications EDD effacent l'affichage à chaque appel de cette fonction.

Structure lexicale

GET_LOCAL_VAR_VALUE(prompt, local_var_name)

Les éléments lexicaux du Builtin GET_LOCAL_VAR_VALUE sont présentés dans le Tableau 141.

Tableau 141 – Builtin GET_LOCAL_VAR_VALUE

Nom du paramètre	Type	Direction	Usage	Description
prompt	char[]	I	m	Message à afficher.
local_var_name	reference d	I	m	Nom de la VARIABLE locale à éditer.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau 367.

4.139 Builtin get_local_var_value

Objet

Le Builtin get_local_var_value affiche le message d'invite spécifié et permet à l'utilisateur d'éditer la valeur d'une VARIABLE locale.

L'invite peut contenir des valeurs de VARIABLE locales et/ou du dispositif incorporées.

NOTE 1 Il n'est pas garanti que les messages précédemment affichés restent à l'écran.

NOTE 2 Certaines applications EDD effacent l'affichage à chaque appel de cette fonction.

Structure lexicale

```
get_local_var_value(prompt, global_var_ids, local_var_name)
```

Les éléments lexicaux du Builtin `get_local_var_value` sont présentés dans le Tableau 142.

Tableau 142 – Builtin `get_local_var_value`

Nom du paramètre	Type	Direction	Usage	Description
prompt	char[]	l	m	Message à afficher.
global_var_ids	array of int	l	o	Matrice qui contient des identifiants uniques d'instance de VARIABLE.
local_var_name	reference d	l	o	Nom de la VARIABLE locale à éditer.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau 367.

4.140 Builtin GET_MANUFACTURER

Objet

Le Builtin `GET_MANUFACTURER` extrait l'équivalent numérique du `MANUFACTURER` à partir de la description du dispositif actuellement utilisée comme indiqué dans l'identification de la description du dispositif avec le mot clé `MANUFACTURER`.

Structure lexicale

```
GET_MANUFACTURER (VOID)
```

L'élément lexical du Builtin `GET_MANUFACTURER` est présenté dans le Tableau 143.

Tableau 143 – Builtin `GET_MANUFACTURER`

Nom du paramètre	Type	Direction	Usage	Description
<return>	int	o	m	MANUFACTURER actuel.

4.141 Builtin `get_more_status`

Objet

Le Builtin `get_more_status` transmet une commande 48 au dispositif et retourne le code de réponse, le statut de communication, et les octets de statut de commande dans la matrice de statut fournie. Il retourne également les octets de données retournés dans la matrice `more_data_info`. Cette fonction traite les octets de statut/données de la même façon que la commande `send`.

La fonction appelante est responsable de l'allocation des matrices pour les données à retourner. Le paramètre `more_data_status` est une matrice de 3 octets, et `more_data_info` est une matrice dont la taille est égale au nombre d'octets de données qui peuvent être retournés par le dispositif dans la commande 48 (maximum 25).

Structure lexicale

```
get_more_status(more_data_status, more_data_info)
```

Les éléments lexicaux du Builtin `get_more_status` sont présentés dans le Tableau 144.

Tableau 144 – Builtin get_more_status

Nom du paramètre	Type	Direction	Usage	Description
more_data_status	char[]	I	m	Spécifie si des informations additionnelles sont disponibles.
more_data_info	char[]	I	m	Spécifie si des informations additionnelles sont disponibles.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau 367.

4.142 Builtin get_resolve_status**Objet**

Le Builtin `get_resolve_status` retourne le numéro d'erreur des Builtin "get_resolve_" les plus récents. Les Builtin de résolution de référence, ceux commençant par "get_resolve_", résolvent l'ID éventuellement dynamique d'un élément de BLOCK_A, PARAMETER, PARAMETER_LIST, VALUE_ARRAY, REFERENCE_ARRAY, COLLECTION, ou RECORD. Si l'un de ces Builtin de résolution de référence rencontre une erreur, il retourne un zéro pour l'ID.

Si la méthode appelante souhaite connaître la nature de l'erreur, elle appelle le Builtin `get_resolve_status` pour obtenir le numéro d'erreur. Ce Builtin est utilisé à des fins de débogage lors de la création de méthodes.

Il retourne les codes pour les erreurs générées par Builtin `resolve_array_ref`, `resolve_block_ref`, `resolve_parm_ref`, `resolve_param_list_ref` ou `resolve_record_ref`.

Structure lexicale

`get_resolve_status(VOID)`

Les éléments lexicaux du Builtin `get_resolve_status` sont présentés dans le Tableau 145.

Tableau 145 – Builtin get_resolve_status

Nom du paramètre	Type	Direction	Usage	Description
<return>	unsigned long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.143 Builtin get_response_code**Objet**

Le Builtin `get_response_code` retourne le code de réponse reçu depuis un dispositif en cas d'appel de communication de Builtin retournant le Builtin d'erreur `fail_on_response_code`. Il retourne également les informations de référence de valeur (`err_id`, `err_member_id`) associées à la demande originale. La demande originale est le dernier appel de Builtin qui a causé une communication.

Un code de réponse est une valeur représentant une condition d'erreur spécifique à une application.

NOTE Une valeur de code de réponse zéro n'est pas considérée comme un code de réponse d'erreur. Chaque code de réponse et type de code de réponse est défini dans l'EDD.

Il obtient le code de réponse retourné lors de la dernière demande de communication, et l'ID d'élément et l'ID de membre pour la valeur consultée lorsque le code de réponse a été retourné.

Structure lexicale

```
get_response_code(resp_code, err_id, err_member_id)
```

Les éléments lexicaux du Builtin `get_response_code` sont présentés dans le Tableau 146.

Tableau 146 – Builtin `get_response_code`

Nom du paramètre	Type	Direction	Usage	Description
resp_code	unsigned long	O	m	Numéro du code de réponse retourné.
err_id	unsigned long	O	m	ID d'élément retourné de l'élément générant le code de réponse.
err_member_id	unsigned long	O	m	ID de membre retourné de l'élément causant un code de réponse.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.144 Builtin `get_response_code_string`

Objet

Le Builtin `get_response_code_string` recherche les codes de réponse de la VARIABLE spécifiée et retourne la description du code de réponse spécifié. Si le code de réponse n'est pas trouvé, et que l'élément est un membre d'un RECORD ou un élément d'un REFERENCE_ARRAY, alors le RECORD ou REFERENCE_ARRAY est contrôlé pour le code spécifié.

Le développeur de méthode appelle ce Builtin si la description d'un code de réponse est inconnue et doit être affichée. Sinon, si le code de réponse est connu, la méthode peut déterminer les mesures à prendre en présence de certaines erreurs. Des descriptions d'erreur non normalisées peuvent également être affichées par une application à la place des codes de réponse normalisés.

Avant d'appeler ce Builtin, la méthode doit utiliser le Builtin `get_response_code` pour obtenir l'item_ID et le member_ID pour l'élément avec un code de réponse. Les ID de paramètre et member_id sont spécifiés en utilisant les Builtin ITEM_ID et MEMBER_ID et le nom ou la référence de la VARIABLE.

Structure lexicale

```
get_response_code_string(id, member_id, code, string, maxlen)
```

Les éléments lexicaux du Builtin `get_response_code_string` sont présentés dans le Tableau 147.

Tableau 147 – Builtin get_response_code_string

Nom du paramètre	Type	Direction	Usage	Description
id	unsigned long	I	m	ID d'élément de l'élément à trouver.
member_id	unsigned long	I	m	ID de membre de l'élément à trouver.
code	unsigned long	I	m	Code de réponse pour lequel la chaîne descriptive est à trouver.
string	char*	O	m	Mémoire tampon pour la description de code de réponse retournée.
maxlen	long	I	m	Longueur maximale de la mémoire tampon.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.145 Builtin get_rspcode_string

Objet

Le Builtin get_rspcode_string retourne la chaîne de code de réponse pour la commande et le code de réponse spécifiés. Si la chaîne est plus longue que la longueur maximale définie dans response_string_length, la chaîne est tronquée. Le code de réponse spécifié doit être valide pour la commande indiquée.

Structure lexicale

get_rspcode_string(cmd_number, code, string, length)

Les éléments lexicaux du Builtin get_rspcode_string sont présentés dans le Tableau 148.

Tableau 148 – Builtin get_rspcode_string

Nom du paramètre	Type	Direction	Usage	Description
cmd_number	int	I	m	Numéro de commande.
code	int	I	m	Code de réponse.
string	DD_STRING	O	m	Mémoire tampon pour la chaîne de code de réponse.
length	int	I		Longueur maximale de la mémoire tampon de chaîne.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau 367.

4.146 Builtin get_rspcode_string_by_id

Objet

Le Builtin get_rspcode_string_by_id retourne une représentation de chaîne d'un response_code donné (retourné par ReadCommand, WriteCommand).

Structure lexicale

get_rspcode_string_by_id(command, response_code, result)

Les éléments lexicaux du Builtin get_rspcode_string_by_id sont présentés dans le Tableau 149.

Tableau 149 – Builtin get_rspcode_string_by_id

Nom du paramètre	Type	Direction	Usage	Description
command	reference	I	m	Commande depuis laquelle le response_code est issu.
response_code	int	I	m	Valeur de response_code (retournée par ReadCommand ou WriteCommand).
result	DD_STRING	O	m	Représentation de chaîne du response_code donné.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau 367.

4.147 Builtin get_signed**Objet**

Le Builtin get_signed extrait la valeur actuelle signée non mise à l'échelle de la VARIABLE traitée.

Il obtient la valeur actuelle d'une VARIABLE signée comme si elle était lue directement depuis un dispositif.

Le Builtin get_signed peut être appelé uniquement par des méthodes associées à une action prédiction, postédition, postlecture ou préécriture. Les méthodes associées aux actions prédiction, postédition, postlecture ou préécriture sont exécutées lorsqu'une VARIABLE pour laquelle une de ces actions est définie en tant qu'attribut est lue par une application, une méthode utilisateur ou une méthode d'édition.

Structure lexicale

get_signed(value)

Les éléments lexicaux du Builtin get_signed sont présentés dans le Tableau 150.

Tableau 150 – Builtin get_signed

Nom du paramètre	Type	Direction	Usage	Description
value	long	I	m	Valeur retournée de la VARIABLE signée.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.148 Builtin get_signed_lelem**Objet**

Le Builtin get_signed_lelem extrait une valeur signée à partir d'une LIST.

Structure lexicale

get_signed_lelem(list_id, index, element_id, subelement_id)

Les éléments lexicaux du Builtin get_signed_lelem sont présentés dans le Tableau 151.

Tableau 151 – Builtin get_signed_lelem

Nom du paramètre	Type	Direction	Usage	Description
list_id	unsigned long	I	m	ID d'élément de la liste.
index	int	I	m	L'index de la liste contenant l'élément à extraire.
element_id	unsigned long	I	m	ID d'élément de l'élément à extraire.
subelement_id	unsigned long	I	m	ID de membre de l'élément à extraire.
<return>	long	O	m	Valeur spécifiée.

4.149 Builtin get_signed_lelem2

Objet

Le Builtin get_signed_lelem2 extrait une valeur signée à partir d'une liste incorporée.

Structure lexicale

```
get_signed_lelem2(list_id, index, embedded_list_id, embedded_list_index,
element_id, subelement_id)
```

Les éléments lexicaux du Builtin get_signed_lelem2 sont présentés dans le Tableau 152.

Tableau 152 – Builtin get_signed_lelem2

Nom du paramètre	Type	Direction	Usage	Description
list_id	unsigned long	I	m	ID d'élément de la liste.
index	int	I	m	L'index de la liste contenant l'élément à extraire.
embedded_list_id	unsigned long	I	m	ID d'élément de la liste incorporée.
embedded_list_index	int	I	m	Index de la liste incorporée contenant l'élément à extraire.
element_id	unsigned long	I	m	ID d'élément de l'élément à extraire.
subelement_id	unsigned long	I	m	ID de membre de l'élément à extraire.
<return>	long	O	m	Valeur spécifiée.

4.150 Builtin get_signed_value

Objet

Le Builtin get_signed_value extrait la valeur actuelle mise à l'échelle de la VARIABLE spécifiée de la mémoire cache ou du dispositif et retourne celle-ci.

L'id et le member_id sont spécifiés en utilisant les Builtin ITEM_ID et MEMBER_ID et le nom ou la référence de la variable. La VARIABLE doit avoir un type de données d'entier signé et doit être une VARIABLE valide dans la mémoire cache ou le dispositif.

Structure lexicale

```
get_signed_value(id, member_id, value)
```

Les éléments lexicaux du Builtin get_signed_value sont présentés dans le Tableau 153.

Tableau 153 – Builtin get_signed_value

Nom du paramètre	Type	Direction	Usage	Description
id	unsigned long	I	m	ID d'élément de la VARIABLE à accéder.
member_id	unsigned long	I	m	ID de membre de la VARIABLE à accéder.
value	long	O	m	Valeur retournée de la VARIABLE signée.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.151 Builtin get_signed_value2**Objet**

Le Builtin get_signed_value2 extrait la valeur actuelle mise à l'échelle de la VARIABLE spécifiée de la mémoire cache ou du dispositif et retourne celle-ci.

L'id et le member_id sont spécifiés en utilisant les Builtin ITEM_ID et MEMBER_ID et le nom ou la référence de la VARIABLE. La VARIABLE doit avoir un type de données d'entier signé et doit être une VARIABLE valide dans la mémoire cache ou le dispositif.

Structure lexicale

```
get_signed_value2(blk_id, blk_num, id, member_id, value)
```

Les éléments lexicaux du Builtin get_signed_value2 sont présentés dans le Tableau 154.

Tableau 154 – Builtin get_signed_value2

Nom du paramètre	Type	Direction	Usage	Description
blk_id	unsigned long	I	m	ID d'élément de l'instance de BLOCK_A à accéder.
blk_num	unsigned long	I	m	Numéro d'occurrence de l'instance de BLOCK_A à accéder.
id	unsigned long	I	m	ID d'élément de la VARIABLE à accéder.
member_id	unsigned long	I	m	ID de membre de la VARIABLE à accéder.
value	long	O	m	Valeur retournée de la VARIABLE signée.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366

4.152 Builtin get_status_code_string**Objet**

Le Builtin get_status_code_string retourne la chaîne de code de statut pour la VARIABLE du dispositif spécifiée et la valeur de code du statut. Si la chaîne est plus longue que la longueur maximale définie dans status_string_length, la chaîne est tronquée. La valeur de code de statut doit être valide pour la VARIABLE du dispositif spécifiée.

Structure lexicale

```
get_status_code_string(variable_name, status_code, status_string, status_string_length)
```

Les éléments lexicaux du Builtin get_status_code_string sont présentés dans le Tableau 155.

Tableau 155 – Builtin get_status_code_string

Nom du paramètre	Type	Direction	Usage	Description
variable_name	char[]	I	m	Nom de la VARIABLE à accéder.
status_code	int	I	m	Valeur de code de statut intéressant.
status_string	char *	O	m	Mémoire tampon de la valeur retournée de la VARIABLE signée.
status_string_length	int	I	m	Taille maximale de la mémoire tampon.
<return>	VOID	—	—	—

4.153 Builtin get_status_string**Objet**

Le Builtin get_status_string recherche les membres de la VARIABLE ENUMERATED ou BIT_ENUMERATED référencée et retourne la description du membre correspondant au statut dans la mémoire tampon spécifiée.

Le Builtin extrait la chaîne associée à une valeur ENUMERATED ou BIT_ENUMERATED. L'utilisation la plus courante de VARIABLE ENUMERATED et BIT_ENUMERATED est l'obtention d'une chaîne associée à un bit de statut défini dans le dispositif pour un statut particulier. Les chaînes associées à des VARIABLES ENUMERATED et BIT_ENUMERATED sont définies dans l'EDDL.

Les paramètres id et member_id sont spécifiés en utilisant les Builtin ITEM_ID et MEMBER_ID et le nom ou la référence de la VARIABLE en question.

Il obtient la description d'un statut spécifique dans un dispositif spécifique.

Structure lexicale

```
get_status_string(id, member_id, status, string, maxlen)
```

Les éléments lexicaux du Builtin get_status_string sont présentés dans le Tableau 156.

Tableau 156 – Builtin get_status_string

Nom du paramètre	Type	Direction	Usage	Description
id	unsigned long	I	m	ID d'élément de la VARIABLE à accéder.
member_id	unsigned long	I	m	ID de membre de la VARIABLE à accéder.
status	unsigned long	I	m	Valeur à rechercher.
string	char*	O	m	Mémoire tampon pour la chaîne retournée.
maxlen	long	I	m	Taille maximale de la mémoire tampon.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.154 Builtin get_stddict_string**Objet**

Le Builtin get_stddict_string prend un identifiant de dictionnaire de texte et retourne la chaîne de texte correspondante dans la mémoire tampon de chaîne fournie. La chaîne retournée est dans la langue correcte pour l'application.

Une entrée du dictionnaire de texte est composée de trois parties: un ID unique, un nom, et une description multilingue. Les chaînes de Builtin sont définies dans le fichier .dct standard.

Structure lexicale

```
get_stdidict_string(id, string, maxlen)
```

Les éléments lexicaux du Builtin `get_stdidict_string` sont présentés dans le Tableau 157.

Tableau 157 – Builtin `get_stdidict_string`

Nom du paramètre	Type	Direction	Usage	Description
id	unsigned long	I	m	Identifiant du dictionnaire de texte.
string	char*	O	m	Mémoire tampon retournée pour la chaîne.
maxlen	long	I	m	Taille maximale de la mémoire tampon.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.155 Builtin `get_string`

Objet

Le Builtin `get_string` extrait la valeur actuelle de chaîne non mise à l'échelle de la VARIABLE traitée.

Il obtient la valeur actuelle de la chaîne VARIABLE comme si elle était lue directement depuis un dispositif. La VARIABLE doit être un paramètre valide. La VARIABLE doit avoir un type de données char (caractère) et être une mémoire tampon d'octet qui peut contenir différents types de données non-ANSI tels que:

- ASCII
- PASSWORD
- EUC (unicode étendu)
- BITSTRING

Les VARIABLES du dispositif de ces types de données doivent être gérées par la méthode parce qu'ils ne sont pas pris en charge par le langage de programmation C. Le Builtin `get_string` est appelé uniquement par des méthodes associées à une action préédition, postédition, postlecture ou préécriture. Les méthodes associées aux actions pré-édition, postédition, postlecture ou postécriture sont exécutées lorsqu'une VARIABLE pour laquelle une de ces actions est définie en tant qu'attribut est lue par une application, une méthode utilisateur ou une méthode d'édition.

Structure lexicale

```
get_string(string, len)
```

Les éléments lexicaux du Builtin `get_string` sont présentés dans le Tableau 158.

Tableau 158 – Builtin `get_string`

Nom du paramètre	Type	Direction	Usage	Description
string	char*	O	m	Mémoire tampon pour la chaîne retournée.
len	long*	I/O	m	Taille maximale de la chaîne. Longueur retournée de la chaîne.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.156 Builtin get_string_lelem

Objet

Le Builtin get_string_lelem extrait une chaîne à partir d'une LIST.

Structure lexicale

```
get_string_lelem(list_id, index, element_id, subelement_id, string, length)
```

Les éléments lexicaux du Builtin get_string_lelem sont présentés dans le Tableau 159.

Tableau 159 – Builtin get_string_lelem

Nom du paramètre	Type	Direction	Usage	Description
list_id	unsigned long	I	m	ID d'élément de la liste.
index	int	I	m	L'index de la liste contenant l'élément à extraire.
element_id	unsigned long	I	m	ID d'élément de l'élément à extraire.
subelement_id	unsigned long	I	m	ID de membre de l'élément à extraire.
string	char*	O	m	Mémoire tampon dans laquelle les données sont à retourner.
length	int*	O	m	Taille des données retournées.
<return>	long	O	m	Valeur spécifiée.

4.157 Builtin get_string_lelem2

Objet

Le Builtin get_string_lelem2 extrait une chaîne à partir d'une liste incorporée.

Structure lexicale

```
get_string_lelem2(list_id, index, embedded_list_id, embedded_list_index, element_id, subelement_id, string, length)
```

Les éléments lexicaux du Builtin get_string_lelem2 sont présentés dans le Tableau 160.

Tableau 160 – Builtin get_string_lelem2

Nom du paramètre	Type	Direction	Usage	Description
list_id	unsigned long	I	m	ID d'élément de la liste.
index	int	I	m	L'index de la liste contenant l'élément à extraire.
embedded_list_id	unsigned long	I	m	ID d'élément de la liste incorporée.
embedded_list_index	int	I	m	Index de la liste incorporée contenant l'élément à extraire.
element_id	unsigned long	I	m	ID d'élément de l'élément à extraire.
subelement_id	unsigned long	I	m	ID de membre de l'élément à extraire.
string	char *	O	m	Mémoire tampon dans laquelle les données sont à retourner.
length	int *	O	m	Taille des données retournées.
<return>	long	O	m	Valeur spécifiée.

4.158 Builtin get_string_value**Objet**

Le Builtin get_string_value extrait la valeur actuelle mise à l'échelle de la VARIABLE spécifiée de la mémoire cache ou du dispositif et retourne celle-ci.

L'id et le member_id sont spécifiés en utilisant les Builtin ITEM_ID et MEMBER_ID et le nom ou la référence de la VARIABLE en question.

Structure lexicale

```
get_string_value(id, member_id, string, len)
```

Les éléments lexicaux du Builtin get_string_value sont présentés dans le Tableau 161.

Tableau 161 – Builtin get_string_value

Nom du paramètre	Type	Direction	Usage	Description
id	unsigned long	I	m	ID d'une référence de VARIABLE.
member_id	unsigned long	I	m	ID de membre d'un BLOCK_A, PARAMETER_LIST, RECORD, VALUE_ARRAY, COLLECTION ou REFERENCE_ARRAY.
string	char*	O	m	Mémoire tampon de la chaîne de résultat.
len	long	O	m	Taille de la chaîne de résultat.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.159 Builtin get_string_value2**Objet**

Le Builtin get_string_value2 extrait la valeur actuelle mise à l'échelle de la VARIABLE spécifiée de la mémoire cache ou du dispositif et retourne celle-ci.

Les paramètres id et member_id sont spécifiés en utilisant les Builtin ITEM_ID et MEMBER_ID et le nom ou la référence de la VARIABLE en question.

Structure lexicale

`get_string_value2(blk_id, blk_num, id, member_id, string, len)`

Les éléments lexicaux du Builtin `get_string_value2` sont présentés dans le Tableau 162.

Tableau 162 – Builtin `get_string_value2`

Nom du paramètre	Type	Direction	Usage	Description
<code>blk_id</code>	unsigned long	I	m	ID d'élément de l'instance de BLOCK_A à accéder.
<code>blk_num</code>	unsigned long	I	m	Numéro d'occurrence de l'instance de BLOCK_A à accéder.
<code>id</code>	unsigned long	I	m	ID d'une référence de VARIABLE.
<code>member_id</code>	unsigned long	I	m	ID de membre d'un BLOCK_A, PARAMETER_LIST, RECORD, VALUE_ARRAY, COLLECTION ou REFERENCE_ARRAY.
<code>string</code>	char*	O	m	Spécifie la mémoire tampon de chaîne de résultat.
<code>len</code>	long	O	m	Taille de la chaîne de résultat.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366

4.160 Builtin GET_TICK_COUNT

Objet

Le Builtin `GET_TICK_COUNT` retourne le temps en ms depuis le dernier démarrage du système. Il peut être utilisé pour les horodatages.

Structure lexicale

`GET_TICK_COUNT(VOID)`

L'élément lexical du Builtin `GET_TICK_COUNT` est présenté dans le Tableau 163.

Tableau 163 – Builtin `GET_TICK_COUNT`

Nom du paramètre	Type	Direction	Usage	Description
<return>	long	O	m	Temps en ms depuis le dernier démarrage du système.

4.161 Builtin get_transfer_status

Objet

La fonction du Builtin `get_transfer_status` permet de demander l'exécution de l'élément EDD en cours de transfert.

Structure lexicale

`get_transfer_status(handle, direction, transfer_status)`

Les éléments lexicaux du Builtin `get_transfer_status` sont présentés dans le Tableau 164.

Tableau 164 – Builtin get_transfer_status

Nom du paramètre	Type	Direction	Usage	Description
handle	unsigned int	I	m	Descripteur du transfert de blocs
Direction	unsigned char	I	m	Spécifie la direction demandée 0x01 pour lecture 0x02 pour écriture 0x03 pour lecture et écriture
transfer_status	unsigned int[4]	O	m	Statut de transfert retourné contenant les informations suivantes: • response code • function code • master-byte-count • device-byte-count
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau 367.

4.162 Builtin get_unsigned**Objet**

Le Builtin get_unsigned extrait la valeur actuelle non signée, non mise à l'échelle, de la VARIABLE traitée.

Le Builtin get_unsigned peut être appelé uniquement par des méthodes associées à une action préédition, postédition, postlecture ou préécriture. Les méthodes associées aux actions pré-édition, postédition, postlecture ou pré-écriture sont exécutées lorsqu'une VARIABLE pour laquelle une de ces actions est définie en tant qu'attribut est lue par une application, une méthode utilisateur ou une méthode d'édition.

Ce Builtin obtient la valeur actuelle d'une variable non signée comme si elle était lue directement depuis un dispositif.

Structure lexicale

get_unsigned(value)

Les éléments lexicaux du Builtin get_unsigned sont présentés dans le Tableau 165.

Tableau 165 – Builtin get_unsigned

Nom du paramètre	Type	Direction	Usage	Description
value	unsigned long	O	m	Valeur retournée.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.163 Builtin get_unsigned_lelem**Objet**

Le Builtin get_unsigned_lelem extrait une valeur non signée à partir d'une LIST.

Structure lexicale

get_unsigned_lelem(list_id, index, element_id, subelement_id)

Les éléments lexicaux du Builtin `get_unsigned_lelem` sont présentés dans le Tableau 166.

Tableau 166 – Builtin `get_unsigned_lelem`

Nom du paramètre	Type	Direction	Usage	Description
<code>list_id</code>	unsigned long	I	m	ID d'élément de la liste.
<code>index</code>	int	I	m	L'index de la liste contenant l'élément à extraire.
<code>element_id</code>	unsigned long	I	m	ID d'élément de l'élément à extraire.
<code>subelement_id</code>	unsigned long	I	m	ID de membre de l'élément à extraire.
<return>	unsigned long	O	m	Valeur spécifiée.

4.164 Builtin `get_unsigned_lelem2`

Objet

Le Builtin `get_unsigned_lelem2` extrait une valeur non signée à partir d'une liste incorporée.

Structure lexicale

```
get_unsigned_lelem2(list_id, index, embedded_list_id, embedded_list_index,
element_id, subelement_id)
```

Les éléments lexicaux du Builtin `get_unsigned_lelem2` sont présentés dans le Tableau 167.

Tableau 167 – Builtin `get_unsigned_lelem2`

Nom du paramètre	Type	Direction	Usage	Description
<code>list_id</code>	unsigned long	I	m	ID d'élément de la liste.
<code>index</code>	int	I	m	L'index de la liste contenant l'élément à extraire.
<code>embedded_list_id</code>	unsigned long	I	m	ID d'élément de la liste incorporée.
<code>embedded_list_index</code>	int	I	m	Index de la liste incorporée contenant l'élément à extraire.
<code>element_id</code>	unsigned long	I	m	ID d'élément de l'élément à extraire.
<code>subelement_id</code>	unsigned long	I	m	ID de membre de l'élément à extraire.
<return>	unsigned long	O	m	Retourne la valeur spécifiée.

4.165 Builtin `get_unsigned_value`

Objet

Le Builtin `get_unsigned_value` extrait la valeur actuelle mise à l'échelle de la VARIABLE spécifiée de la mémoire cache ou du dispositif et retourne celle-ci.

Les paramètres `id` et `member_id` sont spécifiés en utilisant les Builtin `ITEM_ID` et `MEMBER_ID` et le nom ou la référence de la variable en question. La variable doit avoir un type de données non signé. Une valeur non signée peut contenir des variables du dispositif des types de données suivants:

- UNSIGNED
- ENUMERATED
- BIT_ENUMERATED
- INDEX

Les variables du dispositif avec des types de données enumerated, bit_enumerated, et index doivent être gérées par la méthode parce qu'elles ne sont pas prises en charge par le langage de programmation C. La variable doit être un paramètre valide.

Structure lexicale

```
get_unsigned_value(id, member_id, value)
```

Les éléments lexicaux du Builtin get_unsigned_value sont présentés dans le Tableau 168.

Tableau 168 – Builtin get_unsigned_value

Nom du paramètre	Type	Direction	Usage	Description
id	unsigned long	I	m	ID d'élément de la variable à accéder.
member_id	unsigned long	I	m	ID de membre de la variable à accéder.
value	unsigned long	O	m	Valeur retournée de la variable.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.166 Builtin get_unsigned_value2

Objet

Le Builtin get_unsigned_value2 extrait la valeur actuelle mise à l'échelle de la VARIABLE spécifiée de la mémoire cache ou du dispositif et retourne celle-ci.

Les paramètres id et member_id sont spécifiés en utilisant les Builtin ITEM_ID et MEMBER_ID et le nom ou la référence de la variable en question. La variable doit avoir un type de données non signé. Une valeur non signée peut contenir des variables du dispositif des types de données suivants:

- UNSIGNED
- ENUMERATED
- BIT_ENUMERATED
- INDEX

Les variables du dispositif avec des types de données enumerated, bit_enumerated, et index doivent être gérées par la méthode parce qu'elles ne sont pas prises en charge par le langage de programmation C. La variable doit être un paramètre valide.

Structure lexicale

```
get_unsigned_value2(blk_id, blk_num, id, member_id, value)
```

Les éléments lexicaux du Builtin get_unsigned_value2 sont présentés dans le Tableau 169.

Tableau 169 – Builtin get_unsigned_value2

Nom du paramètre	Type	Direction	Usage	Description
blk_id	unsigned long	I	m	ID d'élément de l'instance de BLOCK_A à accéder.
blk_num	unsigned long	I	m	Numéro d'occurrence de l'instance de BLOCK_A à accéder.
id	unsigned long	I	m	ID d'élément de la variable à accéder.
member_id	unsigned long	I	m	ID de membre de la variable à accéder.
value	unsigned long	O	m	Valeur retournée de la variable.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366

4.167 Builtin get_variable_string**Objet**

Le Builtin get_variable_string obtient une représentation textuelle de la VARIABLE référencée.

Structure lexicale

```
get_variable_string(device_var_name, destination_string)
```

Les éléments lexicaux du Builtin get_variable_string sont présentés dans le Tableau 170.

Tableau 170 – Builtin get_variable_string

Nom du paramètre	Type	Direction	Usage	Description
device_var_name	reference	I	m	Nom de la VARIABLE référencée.
destination_string	string	O	m	Mémoire tampon pour la représentation textuelle de la VARIABLE référencée.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau 367.

4.168 Builtin GetCurrentDate**Objet**

Le Builtin GetCurrentDate retourne la meilleure approximation de l'application EDD de la date calendaire actuelle.

Structure lexicale

```
GetCurrentDate()
```

L'élément lexical du Builtin GetCurrentDate est présenté dans le Tableau 171.

Tableau 171 – Builtin GetCurrentDate

Nom du paramètre	Type	Direction	Usage	Description
<return>	time_t	o	m	Date calendaire actuelle.

4.169 Builtin GetCurrentDateAndTime

Objet

Le Builtin GetCurrentDateAndTime retourne la meilleure approximation de l'application EDD de la date et l'heure calendaires actuelles.

Structure lexicale

GetCurrentDateAndTime()

L'élément lexical du Builtin GetCurrentDateAndTime est présenté dans le Tableau 172.

Tableau 172 – Builtin GetCurrentDateAndTime

Nom du paramètre	Type	Direction	Usage	Description
<return>	time_t	o	m	Date et heure calendaires actuelles.

4.170 Builtin GetCurrentTime

Objet

Le Builtin GetCurrentTime détermine l'heure actuelle à partir de la meilleure approximation par l'application hôte de l'heure calendaire actuelle.

Structure lexicale

GetCurrentTime(VOID)

L'élément lexical du Builtin GetCurrentTime est présenté dans le Tableau 173.

Tableau 173 – Builtin GetCurrentTime

Nom du paramètre	Type	Direction	Usage	Description
<return>	time_t	o	m	Heure calendaire actuelle.

4.171 Builtin iassign

Objet

Le Builtin iassign assigne la valeur spécifiée à la variable du dispositif. La variable doit être valide, et doit référencer une variable de type int.

Structure lexicale

iassign(device_var, new_value)

Les éléments lexicaux du Builtin iassign sont présentés dans le Tableau 174.

Tableau 174 – Builtin iassign

Nom du paramètre	Type	Direction	Usage	Description
device_var	reference	l	m	Nom de la variable à accéder.
new_value	int	l	m	Nouvelle valeur de la VARIABLE.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau 367.

4.172 Builtin igetval

Objet

Le Builtin igetval est spécifiquement conçu pour être utilisé dans des méthodes prélecture/écriture et postlecture/écriture. Cette fonction retourne la valeur d'une variable du dispositif comme si elle avait été reçue depuis le dispositif connecté. Des opérations de mise à l'échelle peuvent ensuite être effectuées sur la valeur de variable avant qu'elle ne soit stockée dans l'application EDD.

Structure lexicale

igetval (VOID)

L'élément lexical du Builtin igetval est présenté dans le Tableau 175.

Tableau 175 – Builtin igetval

Nom du paramètre	Type	Direction	Usage	Description
<return>	int	O	m	Valeur de la variable

4.173 Builtin IGNORE_ALL_COMM_STATUS

Objet

Le Builtin IGNORE_ALL_COMM_STATUS efface tous les bits dans les masques de relance et d'interruption du statut de communication. Cela conduit le système à ignorer tous les bits dans la valeur du statut de communication. Le statut de communication est défini comme étant le premier octet de données retourné dans une transaction, lorsque le bit 7 de cet octet est défini.

Les masques de relance et d'interruption sont réinitialisés à leurs valeurs par défaut au début de chaque méthode, de sorte que la nouvelle valeur du masque est valide uniquement pendant la méthode actuelle. Voir le Builtin send_command pour l'implémentation des masques. Voir ABORT_ON_RESPONSE_CODE pour une liste des masques disponibles, et leurs valeurs par défaut.

Le masque affecté par cette fonction est utilisé par les Builtin send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status et display.

Structure lexicale

IGNORE_ALL_COMM_STATUS (VOID)

L'élément lexical du Builtin IGNORE_ALL_COMM_STATUS est présenté dans le Tableau 176.

Tableau 176 – Builtin IGNORE_ALL_COMM_STATUS

Nom du paramètre	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.174 Builtin IGNORE_ALL_DEVICE_STATUS

Objet

Le Builtin IGNORE_ALL_DEVICE_STATUS efface tous les bits dans les masques de relance et d'interruption du statut du dispositif. Cela conduit le système à ignorer tous les bits dans

l'octet de statut du dispositif. Le statut du dispositif est défini comme étant le deuxième octet de données retourné dans une transaction.

Les masques de relance et d'interruption sont réinitialisés à leurs valeurs par défaut au début de chaque méthode, de sorte que la nouvelle valeur du masque est valide uniquement pendant la méthode actuelle. Voir le Builtin `send_command` pour l'implémentation des masques. Voir `ABORT_ON_RESPONSE_CODE` pour une liste des masques disponibles, et leurs valeurs par défaut.

Le masque affecté par cette fonction est utilisé par les Builtin `send`, `send_trans`, `send_command`, `send_command_trans`, `ext_send_command`, `ext_send_command_trans`, `get_more_status` et `display`.

Structure lexicale

`IGNORE_ALL_DEVICE_STATUS (VOID)`

L'élément lexical du Builtin `IGNORE_ALL_DEVICE_STATUS` est présenté dans le Tableau 177.

Tableau 177 – Builtin `IGNORE_ALL_DEVICE_STATUS`

Nom du paramètre	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.175 Builtin `IGNORE_ALL_RESPONSE_CODES`

Objet

Le Builtin `IGNORE_ALL_RESPONSE_CODES` efface tous les bits dans les masques de relance et d'interruption de code de réponse. Cela conduit le système à ignorer toutes les valeurs de code de réponse retournées par le dispositif.

Le code de réponse est défini comme étant le premier octet de données retourné dans une transaction, lorsque le bit 7 de cet octet est 0.

Les masques de relance et d'interruption sont réinitialisés à leurs valeurs par défaut au début de chaque méthode, de sorte que la nouvelle valeur du masque est valide uniquement pendant la méthode actuelle. Voir le Builtin `send_command` pour l'implémentation des masques. Voir le Builtin `ABORT_ON_RESPONSE_CODE` pour une liste des masques disponibles et leurs valeurs par défaut.

Le masque affecté par cette fonction est utilisé par les Builtin `send`, `send_trans`, `send_command`, `send_command_trans`, `ext_send_command`, `ext_send_command_trans`, `get_more_status` et `display`.

Structure lexicale

`IGNORE_ALL_RESPONSE_CODES (VOID)`

L'élément lexical du Builtin `IGNORE_ALL_RESPONSE_CODES` est présenté dans le Tableau 178.

Tableau 178 – Builtin `IGNORE_ALL_RESPONSE_CODES`

Nom du paramètre	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.176 Builtin IGNORE_COMM_ERROR

Objet

Le Builtin IGNORE_COMM_ERROR définit le masque d'erreur de communication de sorte qu'une condition d'erreur de communication est ignorée lors de l'envoi d'une commande.

Les masques de relance et d'interruption sont réinitialisés à leurs valeurs par défaut au début de chaque méthode, de sorte que la nouvelle valeur du masque est valide uniquement pendant la méthode actuelle. Voir la fonction send_command pour l'implémentation des masques. Voir le Builtin ABORT_ON_RESPONSE_CODE pour une liste des masques disponibles et leurs valeurs par défaut.

Le masque affecté par cette fonction est utilisé par les Builtin send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status et display.

Structure lexicale

IGNORE_COMM_ERROR(VOID)

L'élément lexical du Builtin IGNORE_COMM_ERROR est présenté dans le Tableau 179.

Tableau 179 – Builtin IGNORE_COMM_ERROR

Nom du paramètre	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.177 Builtin IGNORE_COMM_STATUS

Objet

Le Builtin IGNORE_COMM_STATUS efface le(s) bit(s) correct(s) dans le masque d'interruption et de relance de statut de communication de sorte que les bits spécifiés dans la valeur comm_status soient ignorés. comm_status est défini comme étant le premier octet de données retourné dans une transaction, lorsque le bit 7 de cet octet est 1.

Les masques de relance et d'interruption sont réinitialisés à leurs valeurs par défaut au début de chaque méthode, de sorte que la nouvelle valeur du masque est valide uniquement pendant la méthode actuelle. Voir la fonction send_command pour l'implémentation des masques. Voir le Builtin ABORT_ON_RESPONSE_CODE pour une liste des masques disponibles et leurs valeurs par défaut.

Le masque affecté par cette fonction est utilisé par les Builtin send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status et display.

Structure lexicale

IGNORE_COMM_STATUS(comm_status)

Les éléments lexicaux du Builtin IGNORE_COMM_STATUS sont présentés dans le Tableau 180.

Tableau 180 – Builtin IGNORE_COMM_STATUS

Nom du paramètre	Type	Direction	Usage	Description
comm_status	int	l	m	Nouveau masque de relance et d'interruption pour ignorer les valeurs.
<return>	VOID	—	—	—

4.178 Builtin IGNORE_DEVICE_STATUS**Objet**

Le Builtin IGNORE_DEVICE_STATUS efface le(s) bit(s) correct(s) dans les masques d'interruption et de relance du statut du dispositif de sorte que les bits spécifiés dans l'octet de statut du dispositif soient ignorés. Le statut du dispositif est défini comme étant le deuxième octet de données retourné dans une transaction.

Les masques de relance et d'interruption sont réinitialisés à leurs valeurs par défaut au début de chaque méthode, de sorte que la nouvelle valeur du masque est valide uniquement pendant la méthode actuelle. Voir la fonction `send_command` pour l'implémentation des masques. Voir le Builtin ABORT_ON_RESPONSE_CODE pour une liste des masques disponibles et leurs valeurs par défaut.

Le masque affecté par cette fonction est utilisé par les Builtin `send`, `send_trans`, `send_command`, `send_command_trans`, `ext_send_command`, `ext_send_command_trans`, `get_more_status` et `display`.

Structure lexicale

IGNORE_DEVICE_STATUS(device_status)

Les éléments lexicaux du Builtin IGNORE_DEVICE_STATUS sont présentés dans le Tableau 181.

Tableau 181 – Builtin IGNORE_DEVICE_STATUS

Nom du paramètre	Type	Direction	Usage	Description
device_status	int	l	m	Nouveau masque de relance et d'interruption pour ignorer les valeurs.
<return>	VOID	—	—	—

4.179 Builtin IGNORE_NO_DEVICE**Objet**

Le Builtin IGNORE_NO_DEVICE configure le masque sans dispositif pour indiquer qu'il convient d'ignorer la condition sans dispositif lors de l'envoi d'une commande.

Les masques de relance et d'interruption sont réinitialisés à leurs valeurs par défaut au début de chaque méthode, de sorte que la nouvelle valeur du masque est valide uniquement pendant la méthode actuelle. Voir la fonction `send_command` pour l'implémentation des masques. Voir le Builtin ABORT_ON_RESPONSE_CODE pour une liste des masques disponibles et leurs valeurs par défaut.

Le masque affecté par cette fonction est utilisé par les Builtin `send`, `send_trans`, `send_command`, `send_command_trans`, `ext_send_command`, `ext_send_command_trans`, `get_more_status` et `display`.

Structure lexicale

IGNORE_NO_DEVICE (VOID)

L'élément lexical du Builtin IGNORE_NO_DEVICE est présenté dans le Tableau 182.

Tableau 182 – Builtin IGNORE_NO_DEVICE

Nom du paramètre	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.180 Builtin IGNORE_RESPONSE_CODE

Objet

Le Builtin IGNORE_RESPONSE_CODE efface le(s) bit(s) correct(s) dans les masques de code de réponse de sorte que la valeur de code de réponse spécifiée soit ignorée. Le code de réponse est défini comme étant le premier octet de données retourné dans une transaction, lorsque le bit 7 de cet octet est 0.

Les masques de relance et d'interruption sont réinitialisés à leurs valeurs par défaut au début de chaque méthode, de sorte que la nouvelle valeur du masque est valide uniquement pendant la méthode actuelle. Voir la fonction send_command pour l'implémentation des masques. Voir le Builtin ABORT_ON_RESPONSE_CODE pour une liste des masques disponibles et leurs valeurs par défaut.

Le masque affecté par cette fonction est utilisé par les Builtin send, send_trans, send_command, send_command_trans, ext_send_command, ext_send_command_trans, get_more_status et display.

Structure lexicale

IGNORE_RESPONSE_CODE (response_code)

Les éléments lexicaux du Builtin IGNORE_RESPONSE_CODE sont présentés dans le Tableau 183.

Tableau 183 – Builtin IGNORE_RESPONSE_CODE

Nom du paramètre	Type	Direction	Usage	Description
response_code	int	l	m	Spécifie le nouveau masque de relance et d'interruption pour ignorer les valeurs
<return>	VOID	—	—	—

4.181 Builtin int_value

Objet

Le Builtin int_value retourne la valeur de la variable du dispositif spécifiée. La variable doit être valide et de type entier.

Structure lexicale

int_value (variable)

Les éléments lexicaux du Builtin int_value sont présentés dans le Tableau 184.

Tableau 184 – Builtin int_value

Nom du paramètre	Type	Direction	Usage	Description
variable	reference	I	m	VARIABLE dont la valeur doit être extraite.
<return>	int	O	m	Valeur de la variable.

4.182 Builtin is_NaN**Objet**

Le Builtin is_NaN vérifie si une valeur à virgule flottante à double précision n'est pas un nombre. "Pas un nombre" (NaN, Not a Number) est spécifié par la norme IEEE 754 comme une valeur double précision non valide.

Structure lexicale

is_NaN(dvalue)

Les éléments lexicaux du Builtin is_NaN sont présentés dans le Tableau 185.

Tableau 185 – Builtin is_NaN

Nom du paramètre	Type	Direction	Usage	Description
dvalue	double	I	m	Nom de la variable.
<return>	long	O	m	1 si dvalue est égal à NaN, sinon zéro.

4.183 Builtin isetval**Objet**

Le Builtin isetval définit la valeur de la variable du dispositif de type INTEGER, ENUMERATED, BIT_ENUMERATED, UNSIGNED_INTEGER and INDEX pour lequel une action pré/post a été désignée à la valeur spécifiée. isetval est spécifiquement conçu pour être utilisé dans des méthodes prélecture/écriture et postlecture/écriture. Cette fonction définit la valeur d'une variable du dispositif à la valeur spécifiée.

Structure lexicale

isetval(value)

Les éléments lexicaux du Builtin isetval sont présentés dans le Tableau 186.

Tableau 186 – Builtin isetval

Nom du paramètre	Type	Direction	Usage	Description
value	int	I	m	Nouvelle valeur de la variable.
<return>	int	O	m	Valeur de la variable.

4.184 Builtin isOffline**Objet**

Le builtin isOffline retourne une valeur non nulle lorsque l'état actuel de l'application hôte (dans le contexte de la méthode appelante) est "Offline".

Structure lexicale

isOffline()

L'élément lexical du Builtin isOffline est présenté dans le Tableau 187.

Tableau 187 – Builtin isOffline

Nom du paramètre	Type	Direction	Usage	Description
<return>	int	O	m	1 si l'application d'EDD ne peut pas communiquer avec le dispositif, 0 si l'application d'EDD peut communiquer avec le dispositif.

4.185 Builtin ITEM_ID

Objet

Le Builtin ITEM_ID peut être utilisé dans des méthodes pour spécifier l'ID d'élément d'une variable pour appeler d'autres méthodes.

Structure lexicale

ITEM_ID(name)

Les éléments lexicaux du Builtin ITEM_ID sont présentés dans le Tableau 188.

Tableau 188 – Builtin ITEM_ID

Nom du paramètre	Type	Direction	Usage	Description
name	reference	I	m	Nom de la variable.
<return>	unsigned long	O	m	ID d'élément de la VARIABLE.

4.186 Builtin itoa (version A)

Objet

Le Builtin convertit une valeur entière en chaîne de texte.

Structure lexicale

itoa(i)

Les éléments lexicaux du Builtin itoa sont présentés dans le Tableau 189.

Tableau 189 – Builtin itoa (version A)

Nom du paramètre	Type	Direction	Usage	Description
i	int	I	m	Valeur entière à convertir en texte.
<return>	char[]	O	m	Texte retourné.

4.187 Builtin itoa (version B)

Objet

Le Builtin itoa convertit une valeur entière en chaîne de texte.

Structure lexicale

```
itoa(i, destination, radix)
```

Les éléments lexicaux du Builtin itoa sont présentés dans le Tableau 190.

Tableau 190 – Builtin itoa (version B)

Nom du paramètre	Type	Direction	Usage	Description
i	int	I	m	Valeur entière à convertir en texte.
destination	char[]	I	m	Chaîne dans laquelle le texte est à stocker.
radix	int	I	m	Base de la valeur (2 = binaire, 8 = octal, 10 = décimal, 16 = hexadécimal).
<return>	char[]	O	m	Texte retourné.

4.188 Builtin ivar_value**Objet**

Le Builtin ivar_value retourne la valeur de la variable spécifiée. La variable source_var_name doit être valide et de type entier.

Structure lexicale

```
ivar_value(source_var_name)
```

Les éléments lexicaux du Builtin ivar_value sont présentés dans le Tableau 191.

Tableau 191 – Builtin ivar_value

Nom du paramètre	Type	Direction	Usage	Description
source_var_name	reference	I	m	Nom de la variable.
<return>	int	O	m	Valeur de la variable.

4.189 Builtin lassign**Objet**

Le Builtin lassign assigne la valeur spécifiée à la variable du dispositif. La variable doit être valide, et doit faire référence à une variable de type entier long.

Structure lexicale

```
lassign(var_name, new_value)
```

Les éléments lexicaux du Builtin lassign sont présentés dans le Tableau 192.

Tableau 192 – Builtin lassign

Nom du paramètre	Type	Direction	Usage	Description
var_name	reference	I	m	Nom de la VARIABLE.
new_value	int	I	m	Nouvelle valeur de la variable.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau 367.

4.190 Builtin Igetval

Objet

Le Builtin Igetval est spécifiquement conçu pour être utilisé dans des méthodes prélecture/écriture et postlecture/écriture. Cette fonction retourne la valeur d'une VARIABLE telle qu'elle a été reçue depuis le dispositif connecté. Des opérations de mise à l'échelle peuvent ensuite être effectuées sur la valeur de VARIABLE avant qu'elle ne soit stockée dans l'application EDD.

Structure lexicale

fgetval (VOID)

L'élément lexical du Builtin Igetval est présenté dans le Tableau 193.

Tableau 193 – Builtin Igetval

Nom du paramètre	Type	Direction	Usage	Description
<return>	int	O	m	Valeur de la variable.

4.191 Builtin ListDeleteElementAt

Objet

Le Builtin ListDeleteElementAt supprime un élément d'une liste. Si la liste est un membre d'un FILE (fichier), l'élément spécifié est supprimé du stockage persistant.

Si un COUNT (compteur) est explicitement déclaré sur la LIST spécifiée, les variables référencées par l'attribut COUNT ne sont pas mises à jour. La méthode est responsable de la mise à jour de COUNT lorsque les éléments sont supprimés de la liste. Si un COUNT n'est pas explicitement déclaré, l'attribut COUNT est automatiquement mis à jour.

Si le paramètre d'index est inférieur à zéro ou supérieur au nombre d'éléments dans la liste, une erreur est retournée. Si le paramètre de liste ne référence pas une instance de LIST, une erreur est retournée.

Structure lexicale

ListDeleteElementAt(list, index)

Les éléments lexicaux du Builtin ListDeleteElementAt sont présentés dans le Tableau 194.

Tableau 194 – Builtin ListDeleteElementAt

Nom du paramètre	Type	Direction	Usage	Description
list	reference	I	m	Nom d'une instance de LIST.
index	int	I	m	Index de l'élément à supprimer.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 367. Les Applications EDD peuvent également retourner le code spécifié dans le Tableau 366 à des fins de diagnostic.

4.192 Builtin ListDeleteElementAt2

Objet

Le Builtin ListDeleteElementAt2 supprime des éléments d'une liste incorporée. Si la liste est un membre d'un FILE, l'élément spécifié est supprimé du stockage persistant.

Structure lexicale

ListDeleteElementAt2(list_id, list_index, embedded_list_id, embedded_index, embedded_count)

Les éléments lexicaux du Builtin ListDeleteElementAt2 sont présentés dans le Tableau 195.

Tableau 195 – Builtin ListDeleteElementAt2

Nom du paramètre	Type	Direction	Usage	Description
list_id	unsigned long	I	m	Élément de la liste.
list_index	int	I	m	Index de la liste contenant la liste incorporée.
embedded_list_id	unsigned long	I	m	ID d'élément de la embedded_list.
embedded_index	int	I	m	Index du premier élément à supprimer.
embedded_count	int	I	m	Nombre d'éléments à supprimer.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 367. Les Applications EDD peuvent également retourner le code spécifié dans le Tableau 366 à des fins de diagnostic.

4.193 Builtin ListInsert

Objet

Le Builtin ListInsert insère un élément dans une liste. Si la liste est un membre d'un FILE, l'élément spécifié est ajouté au stockage persistant.

Si les attributs COUNT et/ou CAPACITY sont explicitement déclarés sur la LIST spécifiée, les variables référencées par ces attributs ne sont pas mises à jour. La méthode est responsable de la mise à jour de COUNT et CAPACITY lorsque des éléments sont insérés dans la liste. Si des attributs COUNT et/ou CAPACITY ne sont pas explicitement déclarés, ces attributs sont automatiquement mis à jour.

Si le paramètre d'index est inférieur à zéro ou supérieur à la capacité de la liste, une erreur est retournée. Si le paramètre de liste ne référence pas une instance de LIST, une erreur est retournée.

Structure lexicale

ListInsert(list, index, item)

Les éléments lexicaux du Builtin ListInsert sont présentés dans l'IEC 61804-3:–, Tableau 150.

Tableau 196 – Builtin ListInsert

Nom du paramètre	Type	Direction	Usage	Description
list	reference	I	m	Nom d'une instance de LIST.
index	int	I	m	Index de l'élément demandé.
item	reference	I	m	Nom de l'élément à insérer.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 367. Les Applications EDD peuvent également retourner le code spécifié dans le Tableau 366 à des fins de diagnostic.

4.194 Builtin ListInsert2

Objet

Le Builtin ListInsert2 insère un élément dans une liste incorporée. Si la liste est un membre d'un FILE (fichier), l'élément spécifié est ajouté au stockage persistant.

Structure lexicale

ListInsert2(list_id, list_index, embedded_list_id, embedded_index, item_id)

Les éléments lexicaux du Builtin ListInsert2 sont présentés dans le Tableau 197.

Tableau 197 – Builtin ListInsert2

Nom du paramètre	Type	Direction	Usage	Description
list_id	unsigned long	I	m	Élément de la liste.
list_index	int	I	m	Index de la liste contenant la liste incorporée.
embedded_list_id	unsigned long	I	m	ID d'élément de la liste incorporée.
embedded_index	int	I	m	Index auquel il convient d'insérer l'élément.
item_id	unsigned long	I	m	ID d'élément de l'élément à insérer.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 367. Les Applications EDD peuvent également retourner le code spécifié dans le Tableau 366 à des fins de diagnostic.

4.195 Builtin log

Objet

Le Builtin log retourne le logarithme naturel d'une valeur à virgule flottante.

Structure lexicale

log(x)

Les éléments lexicaux du Builtin log sont présentés dans le Tableau 198.

Tableau 198 – Builtin log

Nom du paramètre	Type	Direction	Usage	Description
x	double	I	m	Valeur à virgule flottante.
<return>	double	O	m	Logarithme naturel.

4.196 Builtin LOG_MESSAGE**Objet**

Le Builtin LOG_MESSAGE insère un message dans un protocole de journalisation.

Structure lexicale

LOG_MESSAGE(priority, message)

Les éléments lexicaux du Builtin LOG_MESSAGE sont présentés dans le Tableau 199.

Tableau 199 – Builtin LOG_MESSAGE

Nom du paramètre	Type	Direction	Usage	Description
priority	int	I	m	Priorité du message (0 = erreur, 1 = avertissement, 2 = faible).
message	char[]	I	m	Chaîne de message à insérer dans le protocole de journalisation.
<return>	VOID	—	—	

4.197 Builtin log10**Objet**

Le Builtin exp retourne le logarithme base 10 d'une valeur à virgule flottante.

Structure lexicale

log10(x)

Les éléments lexicaux du Builtin log10 sont présentés dans le Tableau 200.

Tableau 200 – Builtin log10

Nom du paramètre	Type	Direction	Usage	Description
x	double	I	m	Valeur à virgule flottante.
<return>	double	O	m	Logarithme base 10.

4.198 Builtin log2**Objet**

Le Builtin log2 retourne le logarithme base 2 d'une valeur à virgule flottante.

Structure lexicale

log2(x)

Les éléments lexicaux du Builtin log2 sont présentés dans le Tableau 201.

Tableau 201 – Builtin log2

Nom du paramètre	Type	Direction	Usage	Description
x	double	I	m	Valeur à virgule flottante.
<return>	double	O	m	Logarithme base 2.

4.199 Builtin long_value**Objet**

Le Builtin long_value retourne la valeur de la variable spécifiée du dispositif. L'identifiant de variable doit être valide et de type long.

Structure lexicale

```
long_value(source_var_name)
```

Les éléments lexicaux du Builtin long_value sont présentés dans le Tableau 202.

Tableau 202 – Builtin long_value

Nom du paramètre	Type	Direction	Usage	Description
source_var_name	reference	I	m	Nom de la variable.
<return>	int	O	m	Valeur de la variable.

4.200 Builtin LongToByte**Objet**

Le Builtin LongToByte génère un vecteur de valeurs de type caractère non signé représentant les éléments particuliers d'une valeur longue.

Structure lexicale

```
LongToByte(in, out1, out2, out3, out4)
```

Les éléments lexicaux du Builtin LongToByte sont présentés dans le Tableau 203.

Tableau 203 – Builtin LongToByte

Nom du paramètre	Type	Direction	Usage	Description
in	long	I	m	Valeur longue à convertir.
out1	unsigned char	O	m	Premier élément de la valeur longue (octet de poids fort).
out2	unsigned char	O	m	Deuxième élément de la valeur longue.
out3	unsigned char	O	m	Troisième élément de la valeur longue.
out4	unsigned char	O	m	Quatrième élément de la valeur longue (octet de poids faible).
<return>	VOID	—	—	—

4.201 Builtin Isetval

Objet

Le Builtin Isetval définit la valeur de la variable du dispositif de type long pour laquelle une action pré/post a été désignée à la valeur spécifiée. Il est spécifiquement conçu pour être utilisé dans des méthodes prélecture/écriture et postlecture/écriture. Cette fonction définit la valeur d'une variable du dispositif à la valeur spécifiée.

Structure lexicale

lsetval(value)

Les éléments lexicaux du Builtin Isetval sont présentés dans le Tableau 204.

Tableau 204 – Builtin Isetval

Nom du paramètre	Type	Direction	Usage	Description
value	int	I	m	Nouvelle valeur de la VARIABLE.
<return>	int	O	m	Valeur de la variable.

4.202 Builtin Ivar_value

Objet

Le Builtin Ivar_value retourne la valeur de la variable du dispositif spécifiée. La variable doit être valide.

Structure lexicale

lvar_value(source_var_name)

Les éléments lexicaux du Builtin Ivar_value sont présentés dans le Tableau 205.

Tableau 205 – Builtin Ivar_value

Nom du paramètre	Type	Direction	Usage	Description
source_var_name	reference	I	m	Nom de la VARIABLE.
<return>	long	O	m	Valeur de la variable.

4.203 Builtin Make_Time

Objet

Le Builtin Make_Time crée une valeur time_t à partir de ses paramètres d'entrée.

Structure lexicale

Make_Time(year, month, day, hour, minute, second, isDST)

Les éléments lexicaux du Builtin Make_Time sont présentés dans le Tableau 206.

Tableau 206 – Builtin Make_Time

Nom du paramètre	Type	Direction	Usage	Description
year	int	I	m	Année.
month	int	I	m	Mois.
day	int	I	m	Jour.
hour	int	I	m	Heure.
minute	int	I	m	Minute.
second	int	I	m	Seconde.
isDST	int	I	m	Spécifie si l'heure d'été est appliquée. S'il est égal à zéro, l'heure d'été n'est pas appliquée; sinon, l'heure d'été est appliquée.
<return>	time_t	O	m	Valeur time_t créée ou -1 si un paramètre non valide est détecté.

4.204 Builtin MEMBER_ID**Objet**

Le Builtin MEMBER_ID peut être utilisé dans des méthodes pour spécifier l'ID de membre d'une variable pour des appels d'autres Builtin. Il retourne l'ID de membre d'un BLOCK, PARAMETER_LIST, RECORD, REFERENCE_ARRAY ou COLLECTION.

Structure lexicale

MEMBER_ID (name)

Les éléments lexicaux du Builtin MEMBER_ID sont présentés dans le Tableau 207.

Tableau 207 – Builtin MEMBER_ID

Nom du paramètre	Type	Direction	Usage	Description
name	reference	I	m	Nom d'un membre d'élément.
<return>	unsigned long	O	m	ID de membre référencé.

4.205 Builtin MenuDisplay**Objet**

Le Builtin MenuDisplay affiche une instance de MENU à l'utilisateur et une liste d'options fournie pour permettre la sélection de l'utilisateur. Une fois qu'une sélection est effectuée, la position de l'option dans la liste d'options est retournée. La position dans la liste est à base zéro. Par exemple, si la liste fournie contient deux éléments et que le premier élément est sélectionné, cette fonction retourne à 0. Si le deuxième élément est sélectionné, un 1 est retourné. Les options sont transmises à la fonction sous la forme d'une chaîne unique délimitée par des points-virgules entre les options. Le MENU doit être traité comme un MENU avec STYLE DIALOG indépendamment du STYLE actuel du MENU.

Structure lexicale

MenuDisplay(menu, options, selection)

Si l'application EDD est incapable d'afficher le menu, une erreur est retournée.

Les éléments lexicaux du Builtin MenuDisplay sont présentés dans le Tableau 208.

Tableau 208 – Builtin MenuDisplay

Nom du paramètre	Type	Direction	Usage	Description
menu	reference	I	m	MENU à afficher.
options	char[]	I	m	Chaîne d'options, constituée de texte délimité, qui définit un ensemble d'options présentées à l'utilisateur.
selection	long*	O	m	Option base zéro sélectionnée (voir les paramètres d'option).
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 367. Les Applications EDD peuvent également retourner le code spécifié dans le Tableau 366 à des fins de diagnostic.

4.206 Builtin method_abort**Objet**

Le Builtin method_abort affiche un message indiquant que la méthode est interrompue et attend un accusé de réception de l'utilisateur.

Après l'accusé de réception, les méthodes d'interruption dans la liste de méthodes d'interruption sont exécutées l'une après l'autre de la tête de la liste (où la première méthode d'interruption a été ajoutée à la liste) à la queue de la liste. La liste de méthodes d'interruption est maintenue uniquement pour l'exécution unique d'une invocation de la méthode.

Une fois que les méthodes d'interruption sont exécutées, la méthode est terminée et le contrôle est retourné à l'application.

Si une méthode d'interruption appelle method_abort, le traitement de la liste de méthodes d'interruption est immédiatement terminé.

Structure lexicale

method_abort(prompt)

Les éléments lexicaux du Builtin method_abort sont présentés dans le Tableau 209.

Tableau 209 – Builtin method_abort

Nom du paramètre	Type	Direction	Usage	Description
prompt	char[]	I	m	Message à afficher.
<return>	VOID	—	—	—

4.207 Builtin nan**Objet**

Le Builtin nan retourne une valeur "not a number" (NaN, pas un nombre) sous la forme d'un double avec un motif binaire fractionnaire basé sur une chaîne. Si la séquence des caractères commence par une lettre de préfixe NaN inactif ou de signalisation, les chiffres hexadécimaux suivants dans la séquence de caractères doivent alors être valides pour ce type de NaN.

La chaîne peut être:

- "" (chaîne vide) génère la valeur NaN par défaut 0x7FF4000000000000.
- "Q" ou "q" est égal à la valeur NaN inactive par défaut.
- "S" ou "s" est égal à la valeur NaN de signalisation par défaut.
- "I_xxxxx" est équivalent au cas où I est "Q", "q", "S", ou "s" suivi par un caractère de soulignement et les chiffres hexadécimaux les plus significatifs de la fraction. Les zéros à droite peuvent être omis.

Structure lexicale

nan(value)

Les éléments lexicaux du Builtin nan sont présentés dans le Tableau 210.

Tableau 210 – Builtin nan

Nom du paramètre	Type	Direction	Usage	Description
value	DD_STRING	I	m	Valeur "Not a number" à retourner.
<return>	double	O	m	Valeur spécifiée.

4.208 Builtin NaN_value

Objet

Le Builtin NaN_value retourne la valeur de "not a number".

Structure lexicale

NaN_value(value)

Les éléments lexicaux du Builtin NaN_value sont présentés dans le Tableau 211.

Tableau 211 – Builtin NaN_value

Nom du paramètre	Type	Direction	Usage	Description
value	double	O	m	Pointeur par l'intermédiaire duquel la valeur est retournée.
<return>	long	O	m	Valeur spécifiée.

4.209 Builtin nanf

Objet

Le Builtin nanf retourne une valeur "not a number" sous la forme d'une valeur à virgule flottante avec un motif binaire fractionnaire basé sur une chaîne. Si la séquence des caractères commence par une lettre de préfixe NaN inactif ou de signalisation, les chiffres hexadécimaux suivants dans la séquence de caractères doivent alors être valides pour ce type de NaN.

La chaîne peut être:

- "" (chaîne vide) génère la valeur NaN par défaut 0x7FA00000.
- "Q" ou "q" est égal à la valeur NaN inactive par défaut.
- "S" ou "s" est égal à la valeur NaN de signalisation par défaut.
- "I_xxxxx" est équivalent au cas où I est "Q", "q", "S", ou "s" suivi par un caractère de soulignement et les chiffres hexadécimaux les plus significatifs de la fraction. Les zéros à droite peuvent être omis.

Structure lexicale

`nanf(value)`

Les éléments lexicaux du Builtin `nanf` sont présentés dans le Tableau 212.

Tableau 212 – Builtin `nanf`

Nom du paramètre	Type	Direction	Usage	Description
value	DD_STRING	I	m	Valeur "Not a number" à retourner.
<return>	float	O	m	Valeur spécifiée.

4.210 Builtin `ObjectReference`

Objet

Le Builtin `ObjectReference` retourne une référence à une instance de `COMPONENT`. La valeur de retour peut être utilisée pour accéder aux objets de données et méthodes d'autres instances de `COMPONENT`.

NOTE Par exemple, dans une EDD d'une station de tête E/S distante, il est possible d'effectuer un contrôle de cohérence avec les modules utilisant `ObjectReference(CHILD)`.

Structure lexicale

`ObjectReference(navigation)`

Les éléments lexicaux du Builtin `ObjectReference` sont présentés dans le Tableau 213.

Tableau 213 – Builtin `ObjectReference`

Nom du paramètre	Type	Direction	Usage	Description
navigation	Une des valeurs suivantes: SELF CHILD FIRST LAST PARENT NEXT PREV	I	m	Pour la direction de navigation, voir l'IEC 61804-3.
<return>	OBJECT_REFERENCE	O	m	Référence à une instance de <code>COMPONENT</code> .

4.211 Builtin `openTransferPort`

Objet

Le Builtin `openTransferPort` ouvre le port de transfert de blocs.

Structure lexicale

`open_transfer_port(port, handle)`

Les éléments lexicaux du Builtin `openTransferPort` sont présentés dans le Tableau 214.

Tableau 214 – Builtin openTransferPort

Nom du paramètre	Type	Direction	Usage	Description
port	unsigned char	I	m	Numéro de port de transfert de blocs.
handle	unsigned int	O	m	Descripteur résultant. Valide uniquement si ce Builtin retourne BI_SUCCESS.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau 367.

4.212 Builtin pop_abort_method

Objet

Le Builtin pop_abort_method enlève une méthode d'interruption du haut de la liste des méthodes d'interruption, qui est la liste des méthodes à exécuter si la méthode actuelle est interrompue.

Structure lexicale

pop_abort_method(VOID)

Les éléments lexicaux du Builtin pop_abort_method sont présentés dans le Tableau 215.

Tableau 215 – Builtin pop_abort_method

Nom du paramètre	Type	Direction	Usage	Description
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau 367

4.213 Builtin pow

Objet

Le Builtin pow retourne la fonction de puissance de deux valeurs à virgule flottante.

Structure lexicale

pow(x, y)

Les éléments lexicaux du Builtin pow sont présentés dans le Tableau 216.

Tableau 216 – Builtin pow

Nom du paramètre	Type	Direction	Usage	Description
x	double	I	m	Base.
y	double	I	m	Exposant.
<return>	double	O	m	Valeur de x élevée à la puissance y.

4.214 Builtin process_abort

Objet

Le Builtin process_abort interrompt la méthode actuelle et exécute les méthodes d'interruption éventuelles, qui sont dans la liste de méthodes d'interruption. Contrairement au Builtin abort, aucun message n'est affiché lorsque cette fonction est exécutée. Cette fonction Builtin peut ne pas être exécutée à partir d'une méthode d'interruption.

Structure lexicale

```
process_abort(VOID)
```

L'élément lexical du Builtin process_abort est présenté dans le Tableau 217.

Tableau 217 – Builtin process_abort

Nom du paramètre	Type	Direction	Usage	Description
<return>	VOID	—	—	—

4.215 Builtin push_abort_method**Objet**

Le Builtin push_abort_method ajoute une méthode d'interruption en haut de la liste des méthodes d'interruption, qui est la liste des méthodes à exécuter si la méthode actuelle est interrompue.

Structure lexicale

```
push_abort_method(abort_method_id)
```

Les éléments lexicaux du Builtin push_abort_method sont présentés dans le Tableau 218.

Tableau 218 – Builtin push_abort_method

Nom du paramètre	Type	Direction	Usage	Description
abort_method_id	unsigned long	I	m	Méthode d'interruption à ajouter.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau 367

4.216 Builtin put_date**Objet**

Le Builtin put_date stocke la nouvelle valeur de date non mise à l'échelle de la variable traitée.

Le Builtin put_date peut être appelé uniquement par des méthodes associées à une action prédiction, postédition, postlecture ou préécriture. Les méthodes associées aux actions prédiction, postédition, postlecture ou préécriture sont exécutées lorsqu'une variable pour laquelle une de ces actions est définie en tant qu'attribut est écrite par une application, une méthode utilisateur ou une méthode d'édition.

Structure lexicale

```
put_date(data, size)
```

Les éléments lexicaux du Builtin put_date sont présentés dans le Tableau 219.

Tableau 219 – Builtin put_date

Nom du paramètre	Type	Direction	Usage	Description
data	char[]	I	m	Nouvelle valeur de la VARIABLE.
size	long	I	m	Longueur de la nouvelle valeur.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.217 Builtin put_date_value

Objet

Le Builtin put_date_value stocke la valeur fournie de la variable mise à l'échelle spécifiée dans la mémoire cache ou dans le dispositif.

Les paramètres id et member_id sont spécifiés en utilisant les Builtin ITEM_ID et MEMBER_ID et le nom ou la référence de la variable en question.

Le Builtin place la valeur spécifiée de la variable date_and_time, time, ou duration dans la mémoire cache ou dans le dispositif.

La variable doit avoir un type de données char (caractère). La chaîne de caractères peut contenir jusqu'à huit caractères et contenir une variable des types de données suivants:

- DATE_AND_TIME
- TIME
- DURATION

La variable doit être valide dans la mémoire cache ou dans le dispositif.

Structure lexicale

put_date_value(id, member_id, data, size)

Les éléments lexicaux du Builtin put_date_value sont présentés dans le Tableau 220.

Tableau 220 – Builtin put_date_value

Nom du paramètre	Type	Direction	Usage	Description
id	unsigned long	I	o	ID d'élément de la VARIABLE à accéder.
member_id	unsigned long	I	o	ID de membre de la VARIABLE à accéder.
data	char[]	I	m	Nouvelle valeur de la VARIABLE.
size	long	I	m	Longueur de la nouvelle valeur.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.218 Builtin put_date_value2

Objet

Le Builtin put_date_value2 stocke la valeur de la variable spécifiée mise à l'échelle dans la mémoire cache ou dans le dispositif.

Les paramètres id et member_id sont spécifiés en utilisant les Builtin ITEM_ID et MEMBER_ID et le nom ou la référence de la variable en question.

Le Builtin place la valeur spécifiée de la variable date_and_time, time, ou duration dans la mémoire cache ou dans le dispositif.

La variable doit avoir un type de données char (caractère). La chaîne de caractères peut contenir jusqu'à huit caractères et contenir une variable des types de données suivants:

- DATA_AND_TIME

- TIME
- DURATION

La variable doit être valide dans la mémoire cache ou dans le dispositif.

Structure lexicale

```
put_date_value2(blk_id, blk_num, id, member_id, data, size)
```

Les éléments lexicaux du Builtin put_date_value2 sont présentés dans le Tableau 221.

Tableau 221 – Builtin put_date_value2

Nom du paramètre	Type	Direction	Usage	Description
blk_id	unsigned long	I	m	ID d'élément de l'instance de BLOCK_A à accéder.
blk_num	unsigned long	I	m	Numéro d'occurrence de l'instance de BLOCK_A à accéder.
id	unsigned long	I	o	ID d'élément de la VARIABLE à accéder.
member_id	unsigned long	I	o	ID de membre de la VARIABLE à accéder.
data	char[]	I	m	Nouvelle valeur de la VARIABLE.
size	long	I	m	Longueur de la nouvelle valeur.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366

4.219 Builtin put_double

Objet

Le Builtin put_double peut être appelé uniquement par des méthodes associées à une action prédiction, postédition, postlecture ou préécriture.

Les méthodes associées aux actions prédiction, postédition, postlecture ou préécriture sont exécutées lorsqu'une variable pour laquelle une de ces actions est définie en tant qu'attribut est écrite par une application, une méthode utilisateur ou une méthode d'édition.

Stocke la nouvelle valeur de la variable double précision telle qu'elle réside dans le dispositif (non mise à l'échelle). Le Builtin put_double stocke la nouvelle valeur double précision non mise à l'échelle de la variable traitée.

Structure lexicale

```
put_double(value)
```

Les éléments lexicaux du Builtin put_double sont présentés dans le Tableau 222.

Tableau 222 – Builtin put_double

Nom du paramètre	Type	Direction	Usage	Description
value	double	I	m	Nouvelle valeur de la VARIABLE.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.220 Builtin put_double_value

Objet

Le Builtin put_double_value stocke la valeur fournie de la variable spécifiée mise à l'échelle dans la mémoire cache ou le dispositif.

Les paramètres id et member_id sont spécifiés en utilisant les Builtin ITEM_ID et MEMBER_ID et le nom ou la référence de la VARIABLE en question.

La valeur spécifiée d'une double variable est placée dans la mémoire cache ou dans le dispositif. La variable doit avoir un type de données double précision et être une variable valide dans la mémoire cache ou dans le dispositif.

Structure lexicale

```
put_double_value(id, member_id, value)
```

Les éléments lexicaux du Builtin put_double_value sont présentés dans le Tableau 223.

Tableau 223 – Builtin put_double_value

Nom du paramètre	Type	Direction	Usage	Description
id	unsigned long	I	o	ID d'élément de la VARIABLE à accéder.
member_id	unsigned long	I	o	ID de membre de la VARIABLE à accéder.
value	double	I	m	Nouvelle valeur de la VARIABLE.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.221 Builtin put_double_value2

Objet

Le Builtin put_double_value2 stocke la valeur fournie de la variable spécifiée mise à l'échelle dans la mémoire cache ou le dispositif.

Les paramètres id et member_id sont spécifiés en utilisant les Builtin ITEM_ID et MEMBER_ID et le nom ou la référence de la VARIABLE en question.

Le Builtin place la valeur spécifiée d'une variable double précision dans la mémoire cache ou dans le dispositif. La variable doit avoir un type de données double précision et être une variable valide dans la mémoire cache ou dans le dispositif.

Structure lexicale

```
put_double_value2(blk_id, blk_num, id, member_id, value)
```

Les éléments lexicaux du Builtin put_double_value2 sont présentés dans le Tableau 224.

Tableau 224 – Builtin put_double_value2

Nom du paramètre	Type	Direction	Usage	Description
blk_id	unsigned long	I	m	ID d'élément de l'instance de BLOCK_A à accéder.
blk_num	unsigned long	I	m	Numéro d'occurrence de l'instance de BLOCK_A à accéder.
id	unsigned long	I	o	ID d'élément de la VARIABLE à accéder.
member_id	unsigned long	I	o	ID de membre de la VARIABLE à accéder.
value	double	I	m	Nouvelle valeur de la VARIABLE.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366

4.222 Builtin put_float**Objet**

Le Builtin put_float stocke la nouvelle valeur à virgule flottante non mise à l'échelle de la variable traitée.

Le Builtin put_float peut être appelé uniquement par des méthodes associées à une action prédiction, postédition, postlecture ou préécriture. Les méthodes associées aux actions prédiction, postédition, postlecture ou préécriture sont exécutées lorsqu'une variable pour laquelle une de ces actions est définie en tant qu'attribut est écrite par une application, une méthode utilisateur ou une méthode d'édition.

Structure lexicale

put_float(value)

Les éléments lexicaux du Builtin put_float sont présentés dans le Tableau 225.

Tableau 225 – Builtin put_float

Nom du paramètre	Type	Direction	Usage	Description
value	double	I	m	Nouvelle valeur de la VARIABLE. La valeur est passée comme un type double précision pour permettre de passer une expression NOTE De nombreux compilateurs réalisent une conversion de simple à double précision avant d'évaluer les expressions.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.223 Builtin put_float_value**Objet**

Le Builtin put_float_value stocke la valeur fournie de la variable mise à l'échelle spécifiée dans la mémoire cache ou dans le dispositif.

Le type de valeur est double précision afin de permettre la promotion automatique qui peut se produire lorsque deux valeurs à virgule flottante sont multipliées.

Les paramètres id et member_id sont spécifiés en utilisant les Builtin ITEM_ID et MEMBER_ID Builtin et le nom ou la référence de la variable en question. La variable doit

avoir un type de données à virgule flottante et être une structure valide dans la mémoire cache ou dans le dispositif.

Structure lexicale

```
put_float_value(id, member_id, value)
```

Les éléments lexicaux du Builtin put_float_value sont présentés dans le Tableau 226.

Tableau 226 – Builtin put_float_value

Nom du paramètre	Type	Direction	Usage	Description
id	unsigned long	I	o	ID d'élément de la VARIABLE à accéder.
member_id	unsigned long	I	o	ID de membre de la VARIABLE à accéder.
value	double	I	m	Nouvelle valeur de la VARIABLE.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.224 Builtin put_float_value2

Objet

Le Builtin put_float_value2 stocke la valeur fournie de la variable mise à l'échelle spécifiée dans la mémoire cache ou dans le dispositif.

Le type de valeur est double précision afin de permettre la promotion automatique qui peut se produire lorsque deux valeurs à virgule flottante sont multipliées.

Les paramètres id et member_id sont spécifiés en utilisant les Builtin ITEM_ID et MEMBER_ID Builtin et le nom ou la référence de la variable en question. La variable doit avoir un type de données à virgule flottante et être une structure valide dans la mémoire cache ou dans le dispositif.

Structure lexicale

```
put_float_value2(blk_id, blk_num, id, member_id, value)
```

Les éléments lexicaux du Builtin put_float_value2 sont présentés dans le Tableau 227.

Tableau 227 – Builtin put_float_value2

Nom du paramètre	Type	Direction	Usage	Description
blk_id	unsigned long	I	m	ID d'élément de l'instance de BLOCK_A à accéder.
blk_num	unsigned long	I	m	Numéro d'occurrence de l'instance de BLOCK_A à accéder.
id	unsigned long	I	o	ID d'élément de la VARIABLE à accéder.
member_id	unsigned long	I	o	ID de membre de la VARIABLE à accéder.
value	double	I	m	Nouvelle valeur de la VARIABLE.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.225 Builtin PUT_MESSAGE

Objet

Le Builtin PUT_MESSAGE affiche le message spécifié à l'écran. Les éventuelles références de VARIABLE locales incorporées sont développées en valeur de variable.

Les références aux variables peuvent être incorporées à la chaîne de message, ce qui permet d'afficher leur valeur en utilisant la syntaxe suivante:

```
%{variable-reference}
```

ou

```
%[format]{variable-reference}
```

Le format facultatif prend en charge toutes les chaînes de format printf décrites dans le langage de programmation C (voir l'ISO/IEC 9899) plus les ajouts présentes dans le Tableau 228.

Tableau 228 – Options de format

Format	Description
D	L'élément référencé est de CLASS DYNAMIC et l'affichage doit être actualisé de manière continue tandis que la fonction Builtin bloque l'exécution de la méthode.
L	La LABEL de l'élément référence doit être affichée.
U	L'unité de l'élément référencé doit être affichée.

Lorsque deux formats ou plus sont donnés, les spécificateurs de format doivent être séparés par une virgule.

NOTE 1 Il n'est pas garanti que les messages précédemment affichés restent à l'écran.

NOTE 2 Certaines applications EDD effacent l'affichage à chaque appel de cette fonction.

Structure lexicale

```
PUT_MESSAGE(message)
```

Les éléments lexicaux du Builtin PUT_MESSAGE sont présentés dans le Tableau 229.

Tableau 229 – Builtin PUT_MESSAGE

Nom du paramètre	Type	Direction	Usage	Description
message	char[]	I	m	Message à afficher.
<return>	VOID	—	—	—

4.226 Builtin put_message

Objet

Le Builtin put_message affiche le message spécifié à l'écran. Les éventuelles références de variable incorporées sont développées en valeur actuelle de la variable. Les variables ENUMERATED sont étendues à leur représentation textuelle plutôt qu'à leur représentation numérique.

La chaîne peut accéder à un nombre quelconque d'identifiants, mais les entrées de matrice ciblées doivent contenir des identifiants de variable valides. Des variables locales et du dispositif peuvent être mélangées dans le message d'affichage.

La syntaxe suivante est utilisée pour référencer une VARIABLE de la liste des VARIABLES donnée dans le paramètre global_var_ids:

```
%{global-var-ids-array-index}
```

ou

```
%[format]{global-var-ids-array-index}
```

Le format facultatif prend en charge toutes les chaînes de format fprintf décrites dans le langage de programmation C (voir l'ISO/IEC 9899) plus les ajouts présentés dans le Tableau 228.

NOTE 1 Il n'est pas garanti que les messages précédemment affichés restent à l'écran.

NOTE 2 Certaines applications EDD effacent l'affichage à chaque appel de cette fonction.

Structure lexicale

```
put_message(message, global_var_ids)
```

Les éléments lexicaux du Builtin put_message sont présentés dans le Tableau 230.

Tableau 230 – Builtin put_message

Nom du paramètre	Type	Direction	Usage	Description
message	char[]	I	m	Message à afficher.
global_var_ids	int[]	I	m	Matrice qui contient des identifiants uniques d'instance de VARIABLE.
<return>	VOID	—	—	—

4.227 Builtin put_signed

Objet

Le Builtin put_signed stocke la nouvelle valeur signée non mise à l'échelle de la variable traitée.

Le Builtin put_signed peut être appelé uniquement par des méthodes associées à une action préédition, postédition, postlecture ou préécriture. Les méthodes associées aux actions pré-édition, postédition, postlecture ou pré-écriture sont exécutées lorsqu'une variable pour laquelle une de ces actions est définie en tant qu'attribut est écrite par une application, une méthode utilisateur ou une méthode d'édition.

Structure lexicale

```
put_signed(value)
```

Les éléments lexicaux du Builtin put_signed sont présentés dans le Tableau 231.

Tableau 231 – Builtin put_signed

Nom du paramètre	Type	Direction	Usage	Description
value	long	I	m	Nouvelle valeur de la VARIABLE.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.228 Builtin put_signed_value

Objet

Le Builtin put_signed_value stocke la nouvelle valeur de la variable mise à l'échelle spécifiée dans la mémoire cache ou dans le dispositif.

Les paramètres id et member_id sont spécifiés en utilisant les Builtin ITEM_ID et MEMBER_ID et le nom ou la référence de la VARIABLE en question.

Place la valeur spécifiée d'une variable de type entier signé dans la mémoire cache ou dans le dispositif.

La variable à écrire doit avoir un type de données long et être une variable valide dans la mémoire cache ou dans le dispositif.

Structure lexicale

```
put_signed_value(id, member_id, value)
```

Les éléments lexicaux du Builtin put_signed_value sont présentés dans le Tableau 232.

Tableau 232 – Builtin put_signed_value

Nom du paramètre	Type	Direction	Usage	Description
id	unsigned long	I	o	ID d'élément de la VARIABLE à accéder.
member_id	unsigned long	I	o	ID de membre de la VARIABLE à accéder.
value	long	I	m	Nouvelle valeur de la VARIABLE.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.229 Builtin put_signed_value2

Objet

Le Builtin put_signed_value2 stocke la nouvelle valeur de la variable mise à l'échelle spécifiée dans la mémoire cache ou dans le dispositif.

Les paramètres id et member_id sont spécifiés en utilisant les Builtin ITEM_ID et MEMBER_ID et le nom ou la référence de la VARIABLE en question.

Place la valeur spécifiée d'une variable de type entier signé dans la mémoire cache ou dans le dispositif.

La variable à écrire doit avoir un type de données long et être une variable valide dans la mémoire cache ou dans le dispositif.

Structure lexicale

```
put_signed_value2(blk_id, blk_num, id, member_id, value)
```

Les éléments lexicaux du Builtin put_signed_value2 sont présentés dans le Tableau 233.

Tableau 233 – Builtin put_signed_value2

Nom du paramètre	Type	Direction	Usage	Description
blk_id	unsigned long	I	m	ID d'élément de l'instance de BLOCK_A à accéder.
blk_num	unsigned long	I	m	Numéro d'occurrence de l'instance de BLOCK_A à accéder.
id	unsigned long	I	o	ID d'élément de la VARIABLE à accéder.
member_id	unsigned long	I	o	ID de membre de la VARIABLE à accéder.
value	long	I	m	Nouvelle valeur de la VARIABLE.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366

4.230 Builtin put_string

Objet

Le Builtin put_string stocke la nouvelle valeur de chaîne de la variable traitée.

La variable doit être valide dans la mémoire cache ou dans le dispositif.

La variable doit avoir un type de données char (caractère); une chaîne de caractères peut contenir des variables du dispositif des types de données suivants:

- ASCII
- PASSWORD
- EUC (unicode étendu)
- BITSTRING

Les variables du dispositif avec les types de données enumerated, bit_enumerated, et index doivent être gérées par la méthode parce qu'elles ne sont pas prises en charge par le langage de programmation C. Le Builtin put_string peut être appelé uniquement par des méthodes associées à une action préédition, postédition, postlecture ou préécriture. Les méthodes associées aux actions préédition, postédition, postlecture ou préécriture sont exécutées lorsqu'une variable pour laquelle une de ces actions est définie en tant qu'attribut est écrite par une application, une méthode utilisateur ou une méthode d'édition.

Structure lexicale

put_string(string, len)

Les éléments lexicaux du Builtin put_string sont présentés dans le Tableau 234.

Tableau 234 – Builtin put_string

Nom du paramètre	Type	Direction	Usage	Description
string	char[]	I	m	Nouvelle valeur de la VARIABLE.
len	long	I	m	Longueur de la nouvelle valeur.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.231 Builtin put_string_value

Objet

Le Builtin put_string_value stocke la valeur fournie de la variable mise à l'échelle spécifiée dans la mémoire cache ou dans le dispositif.

Les paramètres id et member_id sont spécifiés en utilisant les Builtin ITEM_ID et MEMBER_ID et le nom ou la référence de la VARIABLE en question.

La variable doit être valide dans la mémoire cache ou dans le dispositif. La variable doit avoir un type de données char (caractères); une chaîne de caractères peut contenir des variables du dispositif des types de données suivants:

- ASCII
- PASSWORD
- EUC (unicode étendu)
- BITSTRING

Les variables du dispositif avec les types de données enumerated, bit_enumerated, et index doivent être gérées par la méthode parce qu'elles ne sont pas prises en charge par le langage de programmation C.

Structure lexicale

```
put_string_value(id, member_id, string, len)
```

Les éléments lexicaux du Builtin put_string_value sont présentés dans le Tableau 235.

Tableau 235 – Builtin put_string_value

Nom du paramètre	Type	Direction	Usage	Description
id	unsigned long	I	o	ID d'élément de la VARIABLE à accéder.
member_id	unsigned long	I	o	ID de membre de la VARIABLE à accéder.
string	char[]	I	m	Nouvelle valeur de la VARIABLE.
len	long	I	m	Longueur de la nouvelle valeur.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.232 Builtin put_string_value2

Objet

Le Builtin put_string_value2 stocke la valeur fournie de la variable mise à l'échelle spécifiée dans la mémoire cache ou dans le dispositif.

Les paramètres id et member_id sont spécifiés en utilisant les Builtin ITEM_ID et MEMBER_ID et le nom ou la référence de la VARIABLE en question.

La variable doit être valide dans la mémoire cache ou dans le dispositif. La variable doit avoir un type de données char (caractères); une chaîne de caractères peut contenir des variables du dispositif des types de données suivants:

- ASCII
- PASSWORD

- EUC (unicode étendu)
- BITSTRING

Les variables du dispositif avec les types de données enumerated, bit_enumerated, et index doivent être gérées par la méthode parce qu'elles ne sont pas prises en charge par le langage de programmation C.

Structure lexicale

```
put_string_value2(blk_id, blk_num, id, member_id, string, len)
```

Les éléments lexicaux du Builtin put_string_value2 sont présentés dans le Tableau 236.

Tableau 236 – Builtin put_string_value2

Nom du paramètre	Type	Direction	Usage	Description
blk_id	unsigned long	l	m	ID d'élément de l'instance de BLOCK_A à accéder.
blk_num	unsigned long	l	m	Numéro d'occurrence de l'instance de BLOCK_A à accéder.
id	unsigned long	l	o	ID d'élément de la VARIABLE à accéder.
member_id	unsigned long	l	o	ID de membre de la VARIABLE à accéder.
string	char[]	l	m	Nouvelle valeur de la VARIABLE.
len	long	l	m	Longueur de la nouvelle valeur.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366

4.233 Builtin put_unsigned

Objet

Le Builtin put_unsigned stocke la nouvelle valeur non signée, non mise à l'échelle de la variable traitée. La variable doit avoir un type de données non signé et être une variable valide dans la mémoire cache ou le dispositif. Une valeur non signée peut contenir des variables du dispositif des types de données suivants:

- UNSIGNED
- ENUMERATED
- BIT_ENUMERATED
- INDEX

Il convient d'appeler le Builtin put_signed uniquement par des méthodes associées à une action prédiction, postédition, postlecture ou préécriture. Les méthodes associées aux actions prédiction, postédition, postlecture ou préécriture sont exécutées lorsqu'une variable pour laquelle une de ces actions est définie en tant qu'attribut est écrite par une application ou une autre méthode.

Structure lexicale

```
put_unsigned(value)
```

Les éléments lexicaux du Builtin put_unsigned sont présentés dans le Tableau 237.

Tableau 237 – Builtin put_unsigned

Nom du paramètre	Type	Direction	Usage	Description
value	unsigned long	I	o	Nouvelle valeur de la VARIABLE.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.234 Builtin put_unsigned_value**Objet**

Le Builtin put_unsigned_value stocke la valeur fournie de la variable mise à l'échelle spécifiée dans la mémoire cache ou dans le dispositif.

Les paramètres id et member_id sont spécifiés en utilisant les Builtin ITEM_ID et MEMBER_ID et le nom ou la référence de la VARIABLE en question.

La variable doit avoir un type de données non signé et être une variable valide dans la mémoire cache ou le dispositif. Une valeur non signée peut contenir des variables du dispositif des types de données suivants:

- UNSIGNED
- ENUMERATED
- BIT_ENUMERATED
- INDEX

Les variables du dispositif de ces types de données doivent être gérées par la méthode parce qu'elles ne sont pas prises en charge par le langage de programmation C. La variable doit être valide dans la mémoire cache ou dans le dispositif.

Structure lexicale

```
put_unsigned_value(id, member_id, value)
```

Les éléments lexicaux du Builtin put_unsigned_value sont présentés dans le Tableau 238.

Tableau 238 – Builtin put_unsigned_value

Nom du paramètre	Type	Direction	Usage	Description
id	unsigned long	I	o	ID d'élément de la VARIABLE à accéder.
member_id	unsigned long	I	o	ID de membre de la VARIABLE à accéder.
value	unsigned long	I	o	Nouvelle valeur de la VARIABLE.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.235 Builtin put_unsigned_value2**Objet**

Le Builtin put_unsigned_value2 stocke la valeur fournie de la variable mise à l'échelle spécifiée dans la mémoire cache ou dans le dispositif.

Les paramètres `id` et `member_id` sont spécifiés en utilisant les Builtin `ITEM_ID` et `MEMBER_ID` et le nom ou la référence de la VARIABLE en question.

La variable doit avoir un type de données non signé et être une variable valide dans la mémoire cache ou le dispositif. Une valeur non signée peut contenir des variables du dispositif des types de données suivants:

- UNSIGNED
- ENUMERATED
- BIT_ENUMERATED
- INDEX

Les variables du dispositif de ces types de données doivent être gérées par la méthode parce qu'elles ne sont pas prises en charge par le langage de programmation C. La variable doit être valide dans la mémoire cache ou dans le dispositif.

Structure lexicale

```
put_unsigned_value2(blk_id, blk_num, id, member_id, value)
```

Les éléments lexicaux du Builtin `put_unsigned_value2` sont présentés dans le Tableau 239.

Tableau 239 – Builtin `put_unsigned_value2`

Nom du paramètre	Type	Direction	Usage	Description
<code>blk_id</code>	unsigned long	I	m	ID d'élément de l'instance de BLOCK_A à accéder.
<code>blk_num</code>	unsigned long	I	m	Numéro d'occurrence de l'instance de BLOCK_A à accéder.
<code>id</code>	unsigned long	I	o	ID d'élément de la VARIABLE à accéder.
<code>member_id</code>	unsigned long	I	o	ID de membre de la VARIABLE à accéder.
<code>value</code>	unsigned long	I	o	Nouvelle valeur de la VARIABLE.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.236 Builtin `re_read_file`

Objet

Le Builtin `re_read_file` cause la relecture du FILE.

Structure lexicale

```
re_read_file(file_item)
```

Les éléments lexicaux du Builtin `re_read_file` sont présentés dans le Tableau 240.

Tableau 240 – Builtin `re_read_file`

Nom du paramètre	Type	Direction	Usage	Description
<code>file_item</code>	DD_ITEM &	I	m	FILE à lire.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau 367.

4.237 Builtin re_write_file

Objet

Le Builtin re_write_file cause la réécriture du FILE.

Structure lexicale

```
re_write_file(file_item)
```

Les éléments lexicaux du Builtin re_write_file sont présentés dans le Tableau 241.

Tableau 241 – Builtin re_write_file

Nom du paramètre	Type	Direction	Usage	Description
file_item	DD_ITEM &	I	m	FILE à écrire.
<return>	int	O	m	Un des codes de retour spécifiés dans le Tableau 367.

4.238 Builtin read_value

Objet

Le Builtin read_value commande la lecture de la valeur du paramètre depuis le dispositif.

Le Builtin read_value n'effectue aucune action dans une application EDD non en mémoire cache parce qu'une application EDD non en mémoire cache n'a pas de mémoire cache à mettre à jour.

Les paramètres id et member_id sont spécifiés en utilisant les Builtin ITEM_ID et MEMBER_ID et le nom ou la référence de la VARIABLE en question.

Structure lexicale

```
read_value(id, member_id)
```

Les éléments lexicaux du Builtin read_value sont présentés dans le Tableau 242.

Tableau 242 – Builtin read_value

Nom du paramètre	Type	Direction	Usage	Description
id	unsigned long	I	o	ID d'élément de la VARIABLE à accéder.
member_id	unsigned long	I	o	ID de membre de la VARIABLE à accéder.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366.

4.239 Builtin read_value2

Objet

Le Builtin read_value2 cause la lecture de la valeur du paramètre depuis le dispositif.

Le Builtin read_value n'effectue aucune action dans une application EDD non en mémoire cache parce qu'une application EDD non en mémoire cache n'a pas de mémoire cache à mettre à jour.

Les paramètres `id` et `member_id` sont spécifiés en utilisant les Builtin `ITEM_ID` et `MEMBER_ID` et le nom ou la référence de la VARIABLE en question.

Structure lexicale

```
read_value2(blk_id, blk_num, id, member_id)
```

Les éléments lexicaux du Builtin `read_value2` sont présentés dans le Tableau 243.

Tableau 243 – Builtin `read_value2`

Nom du paramètre	Type	Direction	Usage	Description
<code>blk_id</code>	unsigned long	I	m	ID d'élément de l'instance de <code>BLOCK_A</code> à accéder.
<code>blk_num</code>	unsigned long	I	m	Numéro d'occurrence de l'instance de <code>BLOCK_A</code> à accéder.
<code>id</code>	unsigned long	I	o	ID d'élément de la VARIABLE à accéder.
<code>member_id</code>	unsigned long	I	o	ID de membre de la VARIABLE à accéder.
<return>	long	O	m	Un des codes de retour spécifiés dans le Tableau 366

4.240 Builtin ReadCommand

Objet

Le Builtin `ReadCommand` permet de lire des variables avec une commande spécifiée.

Structure lexicale

```
ReadCommand(command_identifiant)
```

Les éléments lexicaux du Builtin `ReadCommand` sont présentés dans le Tableau 244.

Tableau 244 – Builtin `ReadCommand`

Nom du paramètre	Type	Direction	Usage	Description
<code>command_identifiant</code>	reference	I	m	ID de la commande à exécuter.
<return>	long	O	m	Valeur d'erreur spécifique au protocole, dont le codage est défini par le protocole ou dispositif spécifique.

4.241 Builtin readItemFromDevice

Objet

Ce Builtin lit un élément issu d'un transfert de blocs ouvert indiqué par le descripteur.

L'EDD_ITEM peut être un élément de composé, mais ne doit toutefois contenir que des données.

Structure lexicale

```
readItemFromDevice(handle, dd_item)
```

Les éléments lexicaux du Builtin `readItemFromDevice` sont présentés dans le Tableau 245.