BS EN 61804-5:2015

BSI Standards Publication

Function blocks (FB) for process control and electronic device description language (EDDL)

Part 5: EDDL Builtin library

National foreword

This British Standard is the UK implementation of EN 61804-5:2015. It is identical to IEC 61804-5:2015.

The UK participation in its preparation was entrusted to Technical Committee AMT/7, Industrial communications: process measurement and control, including fieldbus.

A list of organizations represented on this committee can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

Published by BSI Standards Limited 2015

ISBN 978 0 580 79625 8
ICS 25.040.40; 35.240.50

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 31 October 2015.

Amendments/corrigenda issued since publication

Date	Text affected

EUROPEAN STANDARD

NORME EUROPÉENNE

EUROPÄISCHE NORM

EN 61804-5

September 2015

ICS 25.040.40; 35.240.50

English Version

Function blocks (FB) for process control and electronic device description language (EDDL) - Part 5: EDDL Builtin library (IEC 61804-5:2015)

Blocs fonctionnels (FB) pour les procédés industriels et le langage de description électronique de produit (EDDL) - Partie 5: Bibliothèque de Builtin EDDL (IEC 61804-5:2015)

Funktionsbausteine für die Prozessautomation und elektronische Gerätebeschreibungssprache - Teil 5: Bibliothek vorgefertigter Unterprogramme (IEC 61804-5:2015)

This European Standard was approved by CENELEC on 2015-07-14. CENELEC members are bound to comply with the CEN/CENELEC Internal Regulations which stipulate the conditions for giving this European Standard the status of a national standard without any alteration.

Up-to-date lists and bibliographical references concerning such national standards may be obtained on application to the CEN-CENELEC Management Centre or to any CENELEC member.

This European Standard exists in three official versions (English, French, German). A version in any other language made by translation under the responsibility of a CENELEC member into its own language and notified to the CEN-CENELEC Management Centre has the same status as the official versions.

CENELEC members are the national electrotechnical committees of Austria, Belgium, Bulgaria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, Former Yugoslav Republic of Macedonia, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Norway, Poland, Portugal, Romania, Slovakia, Slovenia, Spain, Sweden, Switzerland, Turkey and the United Kingdom.

European Committee for Electrotechnical Standardization
Comité Européen de Normalisation Electrotechnique
Europäisches Komitee für Elektrotechnische Normung

CEN-CENELEC Management Centre: Avenue Marnix 17, B-1000 Brussels

Ref. No. EN 61804-5:2015 E

European foreword

The text of document 65E/450/FDIS, future edition 1 of IEC 61804-5, prepared by SC 65E “Devices and integration in enterprise systems” of IEC/TC 65 “Industrial-process measurement, control and automation" was submitted to the IEC-CENELEC parallel vote and approved by CENELEC as EN 61804-5:2015.

The following dates are fixed:

• latest date by which the document has to be implemented at national level by publication of an identical national standard or by endorsement	(dop)	2016-04-14
• latest date by which the national standards conflicting with the document have to be withdrawn	(dow)	2018-07-14

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. CENELEC [and/or CEN] shall not be held responsible for identifying any or all such patent rights.

Endorsement notice

The text of the International Standard IEC 61804-5:2015 was approved by CENELEC as a European Standard without any modification.

Annex ZA
(normative)

Normative references to international publications with their corresponding European publications

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.

NOTE 1 When an International Publication has been modified by common modifications, indicated by (mod), the relevant EN/HD applies.

NOTE 2 Up-to-date information on the latest versions of the European Standards listed in this annex is available here: www.cenelec.eu.

Publication	Year	Title	EN/HD	Year
IEC 61804-3	2015	Function blocks (FB) for process control and EDDL - Part 3: EDDL specification and communication profiles	-	-
IEEE 754	-	IEEE Standard for Binary Floating-Point Arithmetic	-	-

CONTENTS

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.

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.

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 implicitely 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 implicitely 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_acknowlegement
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
writeItemToDevice
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	I	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	I	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	I	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	I	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 browseIdentity

Purpose

The BLOB declaration may not have a declared IDENTITY. The browseIdentity 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 browseIdentity are shown in Table 41.

Table 41 – Builtin browseIdentity

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	I	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_stddict_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	I	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	I	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	I	m	The more recent time.
t0	time_t	I	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	I	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	I	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[]	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.

4.75 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 get_response_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	I	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 VARIBLE 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 modifed.
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 VARIBLE 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	I	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	I	m	The DATE value.
time_value	unsigned long	I	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
blockId	unsigned long	I	m	The item ID of the BLOCK_A instance to be returned.
blockTag	char[]	I	m	The tag of the BLOCK_A instance to be returned.
blk_num	unsigned long	O	m	The occurrence number of the specified BLOCK_A instance.
<return>	long	O	m	One of the return codes specified in Table 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
blockId	unsigned long	I	m	The item ID of the BLOCK_A instances to be counted.
count	unsigned long	O	m	The number of BLOCK_A instances of the specified type that are instantiated in the device.
<return>	long	O	m	One of the return codes specified in Table 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	I	m	The item ID of the list.
index	int	I	m	The index of the list containing the item to be extracted.
element_id	unsigned long	I	m	The item ID of the element to be extracted.
subelement_id	unsigned long	I	m	The member ID of the element to be extracted.
data	char *	O	m	The buffer in which the data is to be returned.
size	int *	O	m	The size of the data returned.
<return>	long	O	m	One of the return codes specified in Table 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
blk_id	unsigned long	I	m	The item ID of the BLOCK_A instance to access.
blk_num	unsigned long	I	m	The occurrence number of the BLOCK_A instance to access.
id	unsigned long	I	m	An ID of a VARIABLE reference.
member_id	unsigned long	I	m	A member ID of a BLOCK_A, PARAMETER_LIST, RECORD, VALUE_ARRAY, COLLECTION or REFERENCE_ARRAY.
data	char[]	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.

4.120 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 DEVICE_REVISION.

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 DEVICE_TYPE.

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_lelem

Purpose

The Builtin get_double_lelem extracts a double value from a LIST.

Lexical structure

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

The lexical element of the Builtin get_double_lelem is shown in Table 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[]	I	m	A message to be displayed.
local_var_name	referenced	I	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[]	I	m	A message to be displayed.
global_var_ids	array of int	I	o	An array which contains unique identifiers of VARIABLE instance.
local_var_name	referenced	I	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
more_data_status	char[]	I	m	Specifies if more info is available.
more_data_info	char[]	I	m	Specifies if more info is available.
<return>	int	O	m	One of the return codes specified in Table 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_parm_ref, resolve_param_list_ref or resolve_record_ref.

Lexical structure

```
get_resolve_status(VOID)
```

The lexical element of the Builtin get_resolve_status is shown in Table 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_stddict_string(id, string, maxlen)
```

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

Table 157 – Builtin get_stddict_string

Parameter name	Type	Direction	Usage	Description
id	unsigned long	I	m	The text dictionary identifier.
string	char*	O	m	The returned buffer for the string.
maxlen	long	I	m	The maximum size of the buffer.
<return>	long	O	m	One of the return codes specified in Table 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-bytecount, • 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 calender 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	I	m	The name of the variable to access.
new_value	int	I	m	The new value of the VARIABLE.
<return>	int	O	m	One of the return codes specified in Table 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	I	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
response_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
variable	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
dvalue	double	I	m	The name of the variable.
<return>	long	O	m	One if the dvalue 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 fist 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

```
log10(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

```
log2(x)
```

The lexical elements of the Builtin log2 are shown in Table 201.

Table 201 – Builtin log2

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_xxxxxx" 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_xxxxxx" 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	I	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[]	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.

4.217 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
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.

4.218 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 asingle 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
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.

4.225 Builtin PUT_MESSAGE

Purpose

The Builtin PUT_MESSAGE will display the specified message on the screen. Any embedded local variable references or references to VARIABLEs 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
blk_id	unsigned long	I	m	The item ID of the BLOCK_A instance to access.
blk_num	unsigned long	I	m	The occurrence number of the BLOCK_A instance to access.
id	unsigned long	I	o	The item ID of the VARIABLE to access.
member_id	unsigned long	I	o	The member ID of the VARIABLE access.
string	char[]	I	m	The new value of the VARIABLE.
len	long	I	m	The length of the new value.
<return>	long	O	m	One of the return codes specified in Table 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
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	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
file_item	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
id	unsigned long	I	m	The item ID of the REFERENCE_ARRAY.
member_id	unsigned long	I	m	The index of the requested REFERENCE_ARRAY element.
<return>	unsigned long	O	m	The item ID of the resolved reference, or zero for an error. In the case of an error, Builtin get_resolve_status may be used to access the actual error code.

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

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

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 Builtins ITEM_ID and MEMBER_ID and the name or reference of the variable in question.

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

Lexical structure

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

The lexical elements of the Builtin resolve_record_ref2 are shown in Table 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	I	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	I	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	I	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 REFRENCE_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 REFRENCE_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
cmd_number	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
cmd_number	int	I	m	The number of command to send.
transaction	int	I	m	The number of the TRANSACTION.
<return>	int	O	m	One of the return codes specified in Table 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 sent_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 Builtin strupr

Purpose

The Builtin strupr returns an upper-case version of a string.

Lexical structure

`strupr(string)`

The lexical elements of the Builtin strupr are shown in Table 313.

Table 313 – Builtin strupr

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 calender 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 1st 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 time_t is copied.
format	DD_STRING	I	m	How the time_t is to be formatted.
timet	time_t	I	m	The time_t to be formatted.
<return>	int	O	m	The number of 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	time_t	I	m	The time_t value to convert.
<return>	unsigned long	O	m	The TIME_VALUE(4) value corresponding to the time of day given by the input parameter timet.

4.320 Builtin timet_to_TIME_VALUE8

Purpose

The Builtin timet_to_TIME_VALUE converts time_t to a TIME_VALUE(8).

Lexical structure

```
timet_to_TIME_VALUE8(timet)
```

The lexical elements of the Builtin timet_to_TIME_VALUE are shown in Table 324.

Table 324 – Builtin timet_to_TIME_VALUE8

Parameter name	Type	Direction	Usage	Description
timet	time_t	I	m	The time_t value to convert.
<return>	unsigned long long	O	m	The TIME_VALUE(8) value corresponding to the date and time given by the input parameter timet.

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 1st, 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 seccond
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 writeItemToDevice

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	I	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	I	m	Zero based index into command 48 data field.
bit	int	I	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	I	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	I	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	I	m	Zero based index into command 48 data field.
Bit	int	I	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	I	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	I	m	The new response code mask.
<return>	VOID	—	—	—

4.351 Builtin XMTR_RETRY_ON_ALL_COMM_STATUS

Purpose

The Builtin XMTR_IGNORE_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	I	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	I	m	Zero based index into command 48 data field.
bit	int	I	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
device_status	INT4	I	m	The new device status retry mask.
<return>	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
<return>	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	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	—

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	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	—

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	Iassign	YES	—
4.172	Igetval	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	Isetval	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_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	—
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	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	—
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	—

This page deliberately left blank

This page deliberately left blank

British Standards Institution (BSI)

BSI is the national body responsible for preparing British Standards and other standards-related publications, information and services.

BSI is incorporated by Royal Charter. British Standards and other standardization products are published by BSI Standards Limited.

About us

We bring together business, industry, government, consumers, innovators and others to shape their combined experience and expertise into standards -based solutions.

The knowledge embodied in our standards has been carefully assembled in a dependable format and refined through our open consultation process. Organizations of all sizes and across all sectors choose standards to help them achieve their goals.

Information on standards

We can provide you with the knowledge that your organization needs to succeed. Find out more about British Standards by visiting our website at bsigroup.com/standards or contacting our Customer Services team or Knowledge Centre.

Buying standards

You can buy and download PDF versions of BSI publications, including British and adopted European and international standards, through our website at bsigroup.com/shop, where hard copies can also be purchased.

If you need international and foreign standards from other Standards Development Organizations, hard copies can be ordered from our Customer Services team.

Subscriptions

Our range of subscription services are designed to make using standards easier for you. For further information on our subscription products go to bsigroup.com/subscriptions.

With **British Standards Online (BSOL)** you'll have instant access to over 55,000 British and adopted European and international standards from your desktop. It's available 24/7 and is refreshed daily so you'll always be up to date.

You can keep in touch with standards developments and receive substantial discounts on the purchase price of standards, both in single copy and subscription format, by becoming a **BSI Subscribing Member**.

PLUS is an updating service exclusive to BSI Subscribing Members. You will automatically receive the latest hard copy of your standards when they're revised or replaced.

To find out more about becoming a BSI Subscribing Member and the benefits of membership, please visit bsigroup.com/shop.

With a **Multi-User Network Licence (MUNL)** you are able to host standards publications on your intranet. Licences can cover as few or as many users as you wish. With updates supplied as soon as they're available, you can be sure your documentation is current. For further information, email bsmusales@bsigroup.com.

BSI Group Headquarters

389 Chiswick High Road London W4 4AL UK

Revisions

Our British Standards and other publications are updated by amendment or revision.

We continually improve the quality of our products and services to benefit your business. If you find an inaccuracy or ambiguity within a British Standard or other BSI publication please inform the Knowledge Centre.

Copyright

Useful Contacts:

Customer Services
Tel: +44 845 086 9001
Email (orders): orders@bsigroup.com
Email (enquiries): cservices@bsigroup.com

Subscriptions
Tel: +44 845 086 9001
Email: subscriptions@bsigroup.com

Knowledge Centre
Tel: +44 20 8996 7004
Email: knowledgecentre@bsigroup.com

Copyright & Licensing
Tel: +44 20 8996 7070
Email: copyright@bsigroup.com

bsi.

...making excellence a habit.™