BS EN 61804-4:2016

BSI Standards Publication

Function blocks (FB) for process control and Electronic Device Description Language (EDDL)

Part 4: EDD interpretation

...making excellence a habit.™

National foreword

This British Standard is the UK implementation of EN 61804-4:2016. It is identical to IEC 61804-4:2015. It supersedes PD CLC/TR 61804-4:2007 which is withdrawn.

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.

ISBN 978 0 580 81632 1
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 29 February 2016.

Amendments/corrigenda issued since publication

Date	Text affected

EUROPEAN STANDARD

NORME EUROPÉENNE

EUROPÄISCHE NORM

EN 61804-4

January 2016

ICS 25.040.40; 35.240.50

Supersedes CLC/TR 61804-4:2007

English Version

Function blocks (FB) for process control and Electronic Device Description Language (EDDL) - Part 4: EDD interpretation (IEC 61804-4:2015)

Blocs fonctionnels (FB) pour les procédés industriels et le langage de description électronique de produit (EDDL) - Partie 4: Interprétation EDD (IEC 61804-4:2015)

Funktionsbausteine für die Prozessautomation und elektronische Gerätebeschreibungssprache - Teil 4: Interpretation von Gerätebeschreibungen (IEC 61804-4:2015)

CENELEC

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

European foreword

The text of document 65E/465/FDIS, future edition 1 of IEC 61804-4, 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 61804-4:2016.

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)	2016-08-11
• latest date by which the national standards conflicting with the document have to be withdrawn	(dow)	2018-11-11

This document supersedes CLC/TR 61804-4:2007.

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 61804-4:2015 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 61784-1	-	Industrial communication networks - Profiles -- Part 1: Fieldbus profiles	EN 61784-1	-
IEC 61784-2	-	Industrial communication networks - Profiles - Part 2: Additional fieldbus profiles for real-time networks based on ISO/IEC 8802-3	EN 61784-2	-
IEC 61804-2	-	Function Blocks (FB) for process control -- Part 2: Specification of FB concept	EN 61804-2	-
IEC 61804-3	-	Function blocks (FB) for process control and EDDL - Part 3: EDDL specification and communication profiles	EN 61804-3	-
IEC 61804-5	-	Function blocks (FB) for process control and EDDL - Part 5: EDDL Builtin library	EN 61804-5	-
ISO/IEC 10918	series	Information technology_- Digital compression and coding of continuous-tone still images: JPEG File Interchange Format (JFIF)	-	series
ISO/IEC 15948	-	Information technology - Computer graphics and image processing - Portable Network Graphics (PNG) - Functional specification	-	-

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

____

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

Part 4: EDD interpretation

FOREWORD

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

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

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

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

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

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

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

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

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

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

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

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

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

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

INTRODUCTION

This part of IEC 61804 was developed using material from FDI Cooperation LLC (Foundation™ Fieldbus[1], HART®[2] Communication Foundation (HCF), PROFIBUS™[3] Nutzerorganisation e.V. (PNO)), OPC Foundation (OPCF) and FDT Group. IEC 61804 has the general title "Function blocks (FB) for process control and Electronic Device Description Language (EDDL)".

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

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

This part of IEC 61804

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

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

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

---

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

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

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

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

Part 4: EDD interpretation

1 Scope

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

2 Normative references

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

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

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

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

IEC 61804-3[4], *Function blocks (FB) for process control and Electronic device description language (EDDL) – Part 3: EDDL syntax and semantics*

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

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

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

3 Terms, definitions, abbreviated terms, acronyms and conventions

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

4 To be published.

5 To be published.

3.1 General terms and definitions

3.1.1
EDD developer
individual or team that develops an EDD

3.1.2
container
user interface elements that contain other user interface elements

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

3.1.3
contained item
user interface elements that can be contained in containers

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

3.1.4
device developer
individual or team that develops a device and an EDD that describes the device

3.1.5
handheld
device with limited display resolution that restricts EDD applications user interface

3.2 Terms and definitions related to modular devices

3.2.1
channel
connection to a process that is being measured or controlled

3.2.2
component
software or hardware item contained within the modular device concept

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

3.2.3
interface
basic declarations of basic constructs

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

3.2.4
modal window
a child window that requires users to interact with it before they can return to operating the parent application, thus preventing the workflow on the application main window

3.2.5

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

3.3 Abbreviated terms and acronyms

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

3.4 Conventions

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

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

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

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

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

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

4 EDDL user interface description

4.1 Overview

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

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

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

4.2 Menu conventions for handheld applications

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

Table 1 – List of defined root menu identifiers for handhelds

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

4.3 Menu conventions for PC-based applications

4.3.1 Overview

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

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

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

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

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

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

4.3.2 Online Root Menus

4.3.2.1 General

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

Table 3 – Fall back alternatives for online root menus

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

4.3.2.2 Diagnostic Root Menu

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

4.3.2.3 Process variable Root Menu

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

4.3.2.4 Device Root Menu

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

4.3.2.5 Maintenance Root Menu

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

4.3.3 Offline Root Menu

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

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

Table 4 – Fall back alternatives for offline root menus

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

4.3.4 Example of EDD menu structure

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

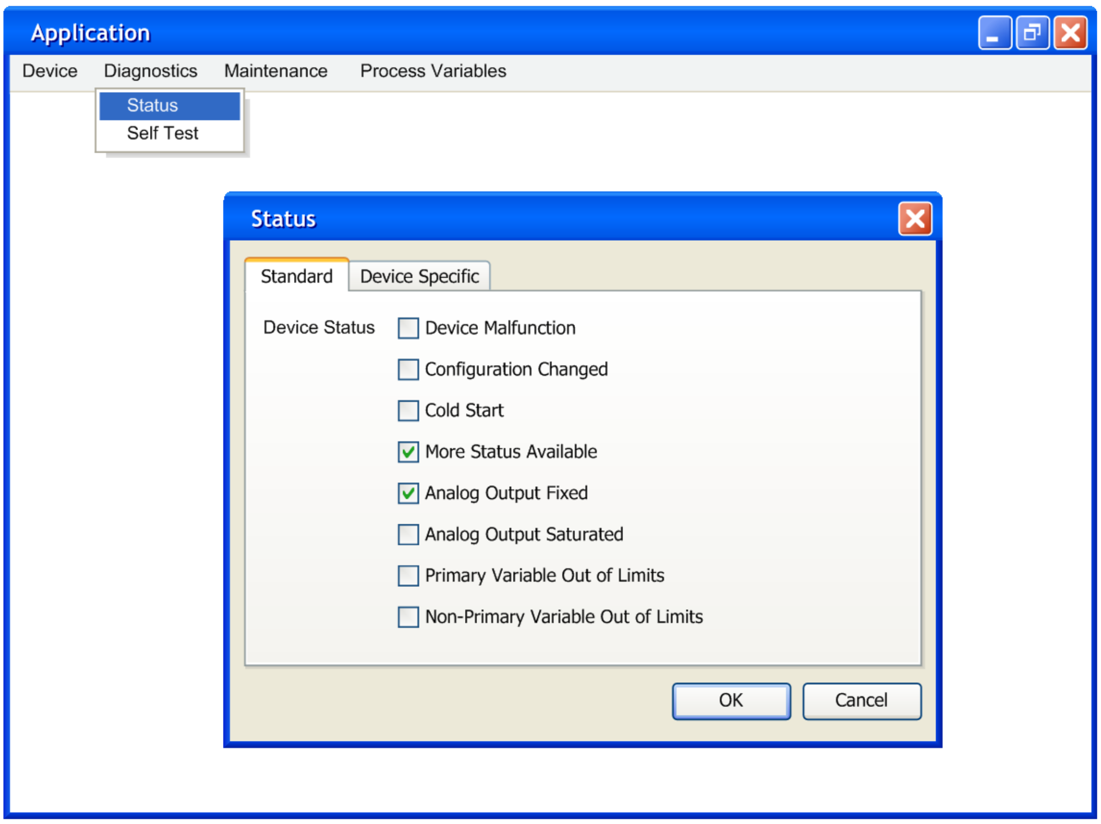
```
MENU diagnostic_root_menu
{
    LABEL "Diagnostics";
    STYLE MENU;                         /* not required to define STYLE in this case, */
                                        /* because of default STYLE of the root menu */
    ITEMS
    {
        status_window,                  /* menu: style=window */
        self_test                       /* method */
    }
}

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

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

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

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

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

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

MENU mode_page
{
    LABEL "Process Variables";
    STYLE PAGE;
    ITEMS
    {
        transducer_group,           /* menu: style=group */
        function_group              /* menu: style=group */
    }
}

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

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

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

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

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

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

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

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

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

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

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

CHART pressure_chart
{
    /* elided */
}

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

CHART temperature_chart
{
    /* elided */
}

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

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

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

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


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

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

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

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

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

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

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

Figure 1 – EDD example of root menus

4.3.5 User interface

4.3.5.1 Diagnostics

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

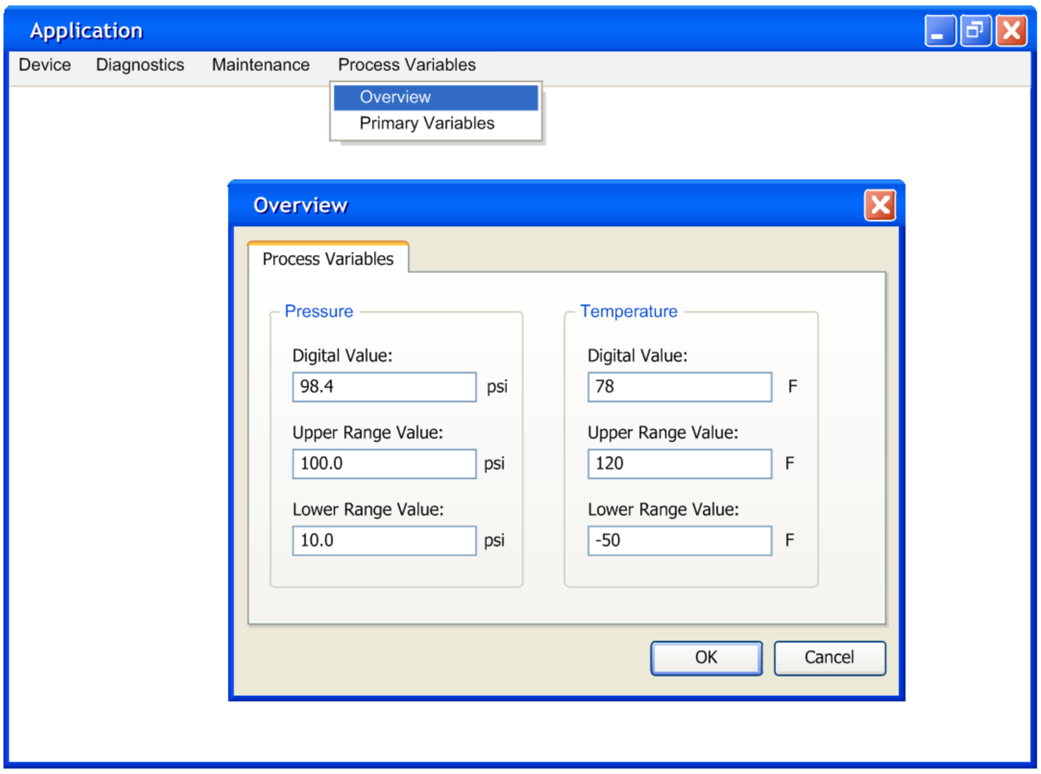


Device Status
- [ ] Device Malfunction
- [ ] Configuration Changed
- [ ] Cold Start
- [✓] More Status Available
- [✓] Analog Output Fixed
- [ ] Analog Output Saturated
- [ ] Primary Variable Out of Limits
- [ ] Non-Primary Variable Out of Limits

Figure 2 – Example of an EDD application for diagnostics

4.3.5.2 Process variables

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



Pressure
Digital Value: 98.4 psi
Upper Range Value: 100.0 psi
Lower Range Value: 10.0 psi

Temperature
Digital Value: 78 F
Upper Range Value: 120 F
Lower Range Value: -50 F

Figure 3 – Example of an EDD application for process variables

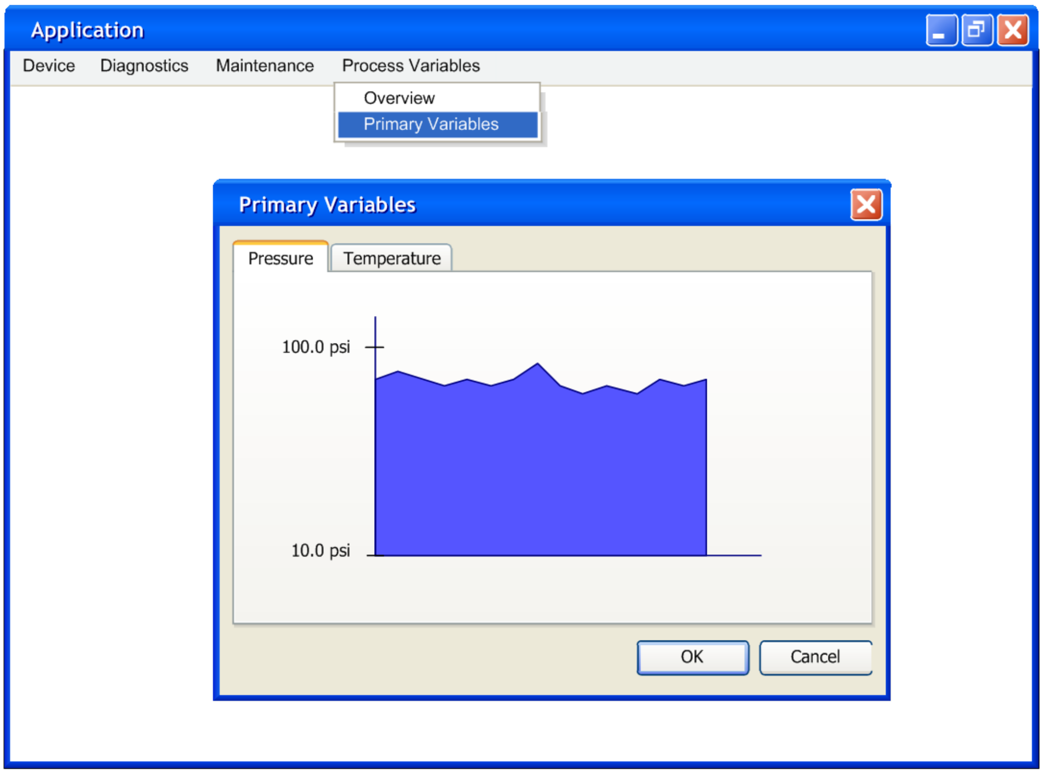
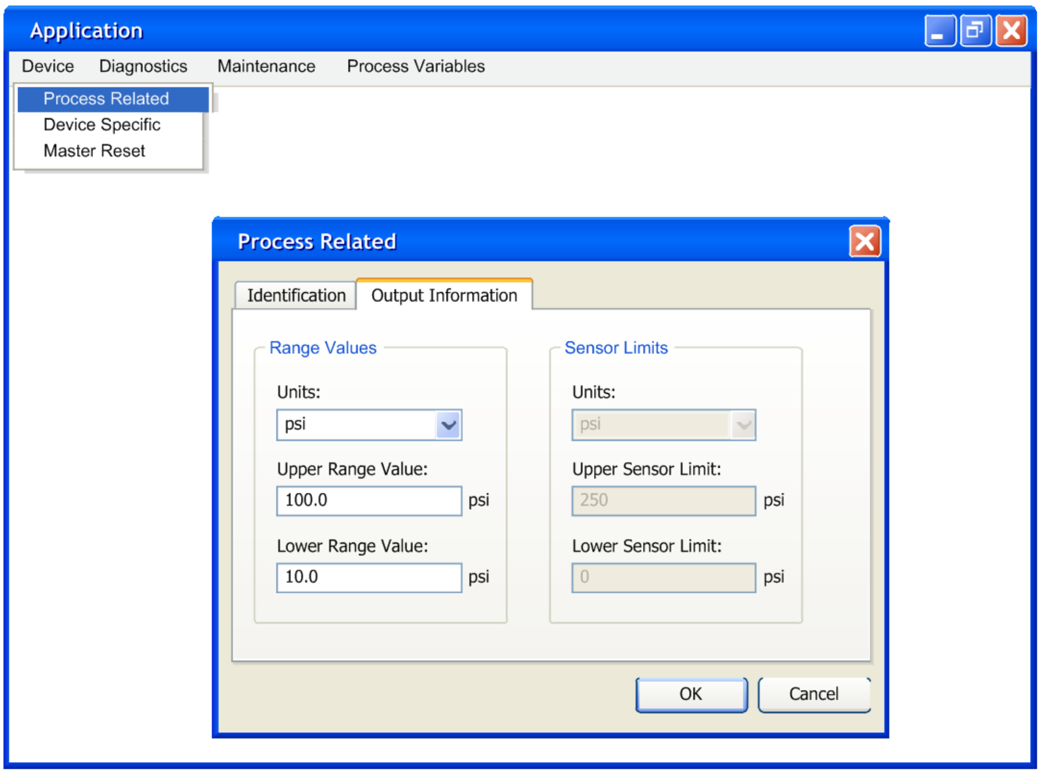


Figure 4 – Example of an EDD application for primary variables

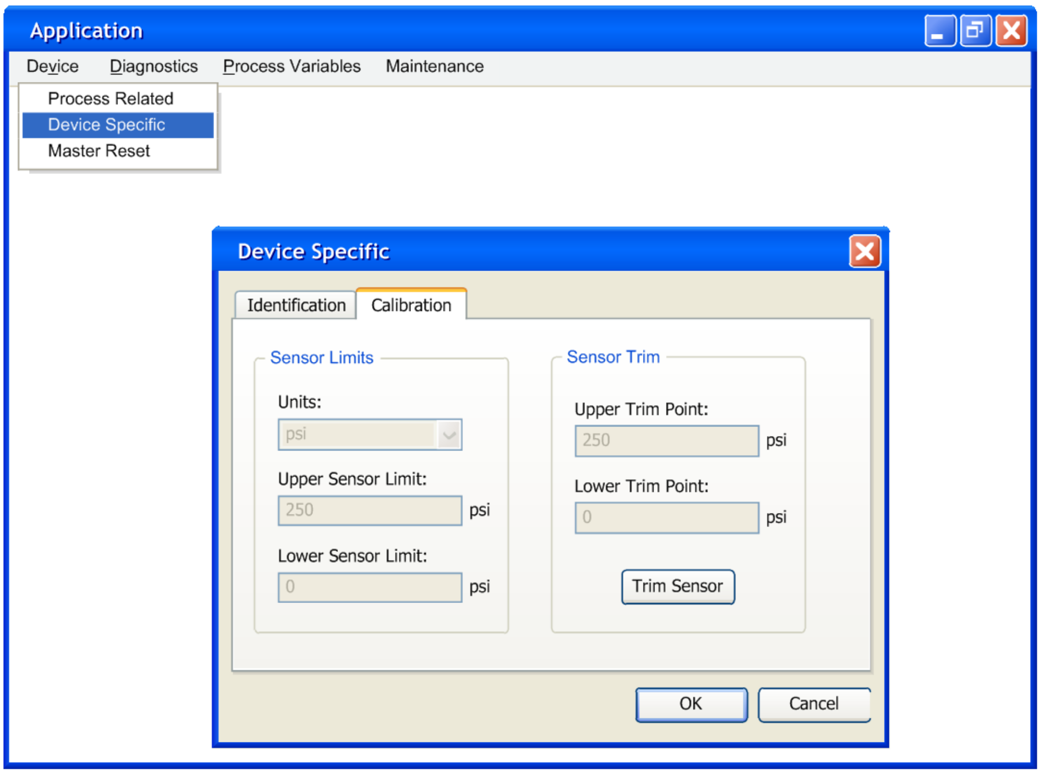
4.3.5.3 Device features

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



Units: psi
Upper Range Value: 100.0 psi
Lower Range Value: 10.0 psi
Units: psi
Upper Sensor Limit: 250 psi
Lower Sensor Limit: 0 psi

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



Units: psi
Upper Sensor Limit: 250 psi
Lower Sensor Limit: 0 psi
Upper Trim Point: 250 psi
Lower Trim Point: 0 psi

Figure 6 – Example of an EDD application for device features

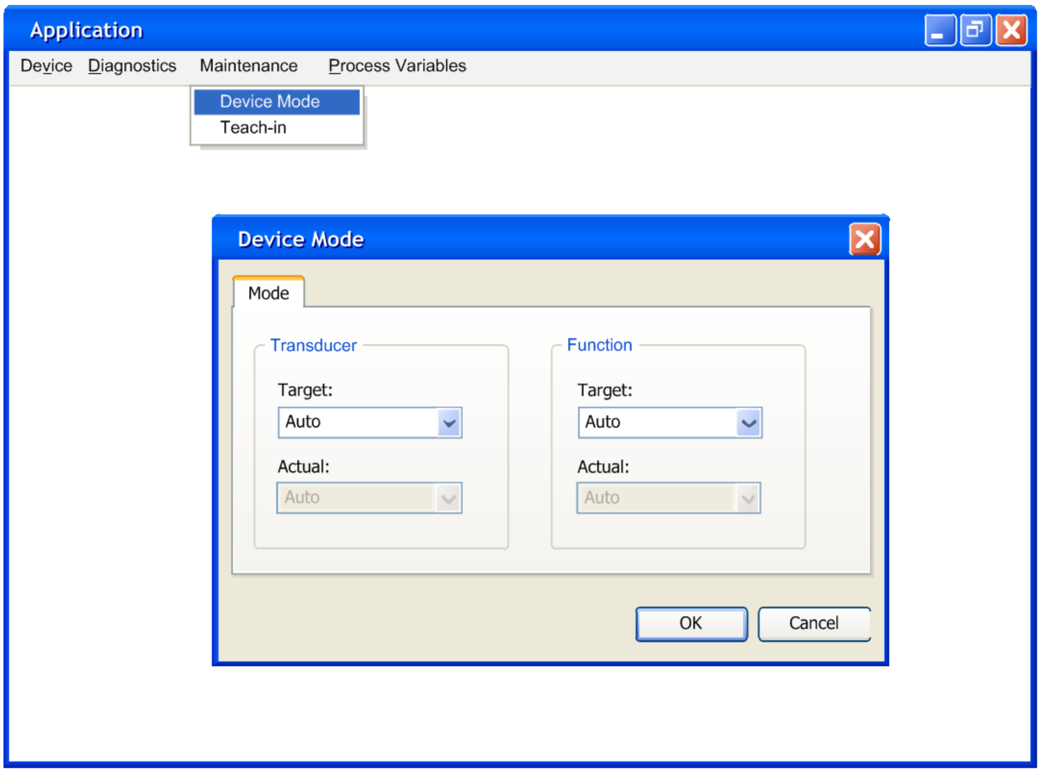


Figure 7 – Example of an EDD application for maintenance features

4.4 Containers and contained items

4.4.1 Overview

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

- containers; and
- contained items.

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

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

4.4.2 Containers

4.4.2.1 Permitted and default STYLEs

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

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

General rules:

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

Table 5 – Permitted contained items and default STYLES

Container	**Permitted contained items**															**Default STYLE of contained items, used if STYLE is not defined**
	MENU	WINDOW, DIALOG	PAGE	GROUP	TABLE	EDIT_DISPLAY	VARIABLE	COLLECTION, RECORD	ARRAY, LIST	IMAGE, CHART, GRAPH, GRID	static text	METHOD	PLUGIN	COLUMNBREAK, ROWBREAK	SEPARATOR	
MENU	X	X	D	D	2	X	7	2	2	2	2	X	X	6	6	MENU of STYLE MENU, if no data item is in the menu. DIALOG, if any data item is in the menu.
DIALOG, WINDOW	X	4	X	X	X	4	X	X	X	X	X	4	4	X	8	PAGE
PAGE	X	4	D	X	X	4	X	X	X	X	X	4	4	X	3	GROUP
GROUP	X	4	D	1	D	4	X	1	X	X	X	4	4	X	3	GROUP if the parent of the parent GROUP is not of STYLE GROUP. DIALOG rendered as button if the parent of the parent GROUP is of STYLE GROUP.
TABLE	4	D	D	D	X	4	X	X	X	5	X	4	4	6	6	TABLE

Key

X Allowed.

D Default STYLE shall be used (see column “Default STYLE of submenu if not defined”).

1 Only one sublevel is allowed e.g. GROUP may contain a GROUP but this GROUP shall not contain GROUPs, COLLECTIONs or RECORDs.

2 Shall be rendered in a dialog.

3 Shall be interpreted as COLUMNBREAK.

4 Shall be rendered as button or hyperlink.

5 Shall be rendered as button, hyperlink or inline.

6 Shall be rendered as a line or ignored.

7 Shall be rendered as button, hyperlink or ignored.

8 SEPARATOR shall be treated as COLUMNBREAK.

4.4.2.2 Menu

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

4.4.2.3 Window

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

4.4.2.4 Dialog

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

4.4.2.5 Table

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

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

4.4.2.6 Page

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

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

COLUMNBREAKs and SEPARATORs between pages shall be ignored.

4.4.2.7 Group

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

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

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

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

```
VARIABLE ai1_strev
{
    LABEL "Analog Input 1: Static Revision";
    TYPE INTEGER(4);
}

VARIABLE ai1_blomo
{
    LABEL "Analog Input 1: Block Mode";
    TYPE INTEGER(1);
}

MENU ai1_group_double                        /* Leads to double display of */
{
    LABEL "Analog Input 1";
    STYLE GROUP;
    ITEMS
    {
        ai1_strev,
        ai1_blomo
    }
}

COLLECTION ai1_groupcol
{
    MEMBERS
    {
        strev,ai1_strev,"Static Revision";
        blomo,ai1_blomo,"Block Mode";
    }
}

MENU ai1_group_single                        /* Leads to single display of */
{
    LABEL "Analog Input 1";
    STYLE GROUP;
    ITEMS
    {
        ai1_groupcol.strev,
        ai1_groupcol.blomo
    }
}
```

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

4.4.3 Contained items

4.4.3.1 Overview

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

4.4.3.2 Methods

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

4.4.3.3 Variables

4.4.3.3.1 General

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

The general handling of variables are as follows.

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

4.4.3.3.2 Variable of TYPE BIT_ENUMERATED

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

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

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

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

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

Figure 9 – Displaying single bits of BIT_ENUMERATED

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

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

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

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

Figure 10 – Displaying multiple bits of BIT_ENUMERATED

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

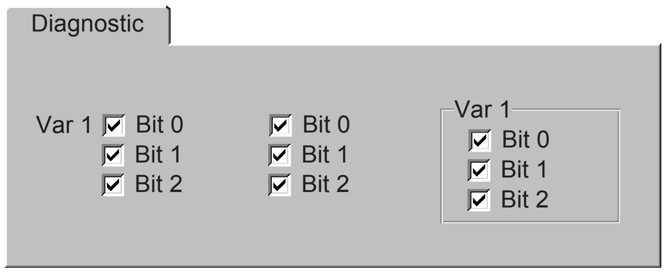


Figure 11 – Example of an EDD application for a variable of type BIT_ENUMERATED

4.4.3.3.3 Variable of TYPE INDEX

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

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

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

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

4.4.3.4 Records and collections

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

4.4.3.5 Arrays and lists

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

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

4.4.3.6 Edit displays

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

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

4.4.3.7 Graphs

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

4.4.3.8 Charts

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

4.4.3.9 Images

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

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

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

The EDD application shall not scale up an IMAGE.

The EDD application shall keep original aspect ratio of IMAGEs.

4.4.3.10 Static text

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

4.4.3.11 Grid

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

4.5 Layout rules

4.5.1 Overview

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

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

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

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

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

4.5.2 Layout rules for WIDTH and HEIGHT

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

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

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

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

Table 6 – WIDTH and HEIGHT span and applicability

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

4.5.3 Layout rules for COLUMNBREAK and ROWBREAK

4.5.3.1 Layout for protruding elements

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

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

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

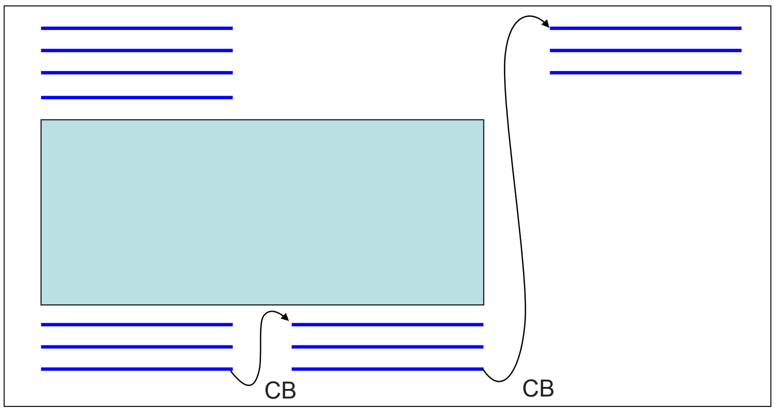


Figure 13 – Layout for protruding elements

4.5.3.2 Layout for partially filled rows

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

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

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

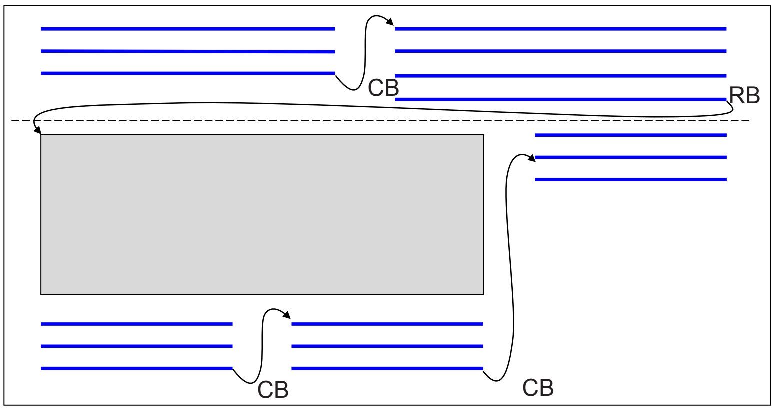


Figure 15 – Layout for partially filled rows

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

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

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

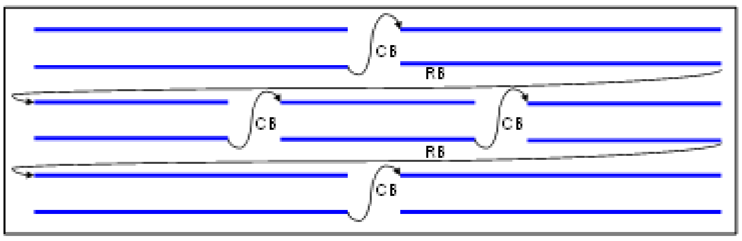


Figure 17 – Layout for partially filled rows

4.5.3.3 Layout for oversized elements

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

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

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

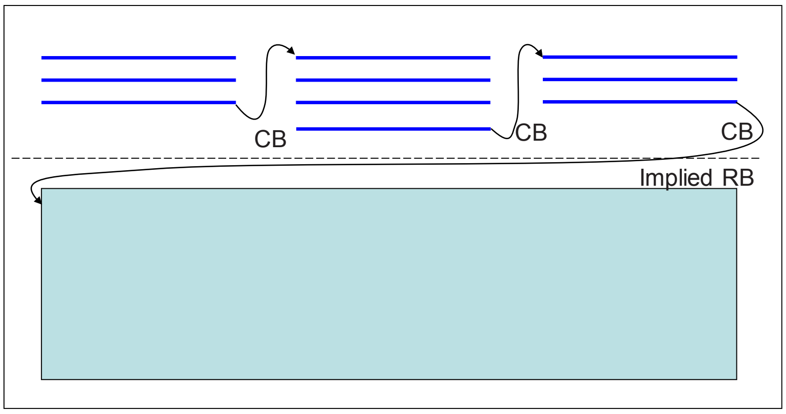


Figure 19 – Layout for oversized elements

4.5.3.4 Layout for columns in stacked group

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

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

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

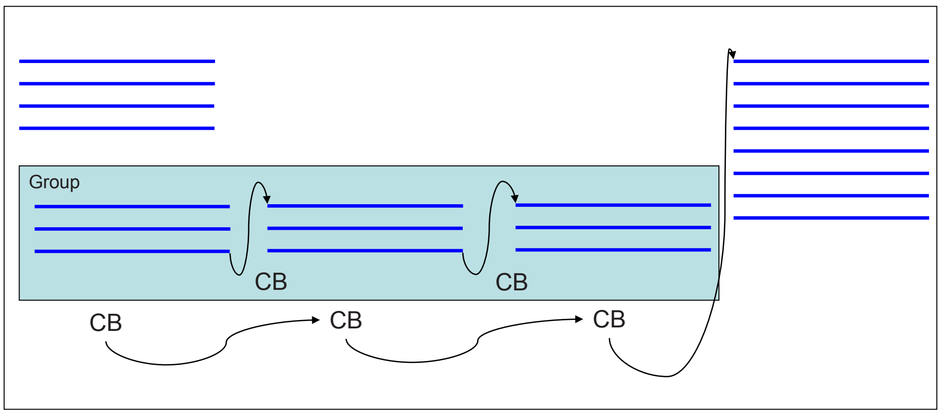


Figure 21 – Layout for columns in stacked group

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

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

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

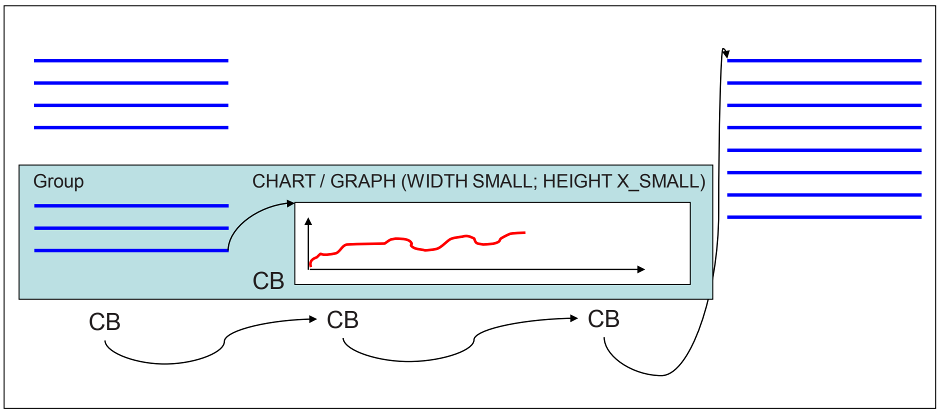


Figure 23 – Layout for columns with GRAPHs in stacked group

4.5.4 Layout examples

4.5.4.1 Single-column layout

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

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

Figure 24 – Example of an EDD for an overview menu

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

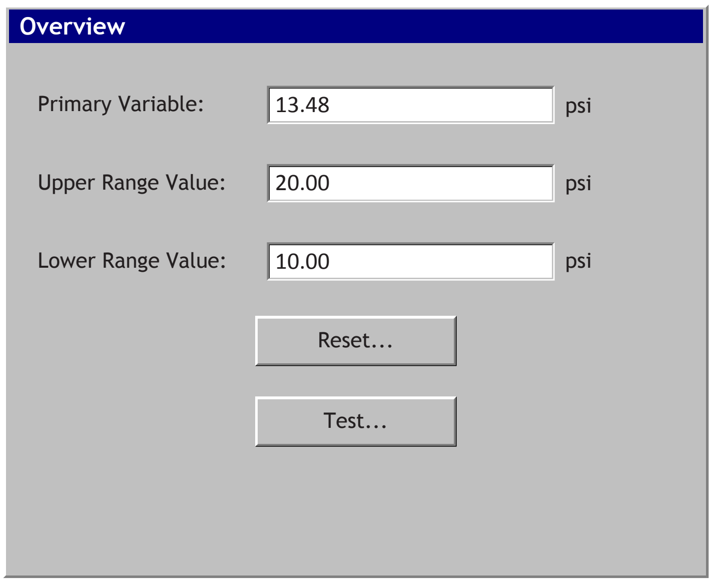


Figure 25 – Example of an EDD application for an overview window

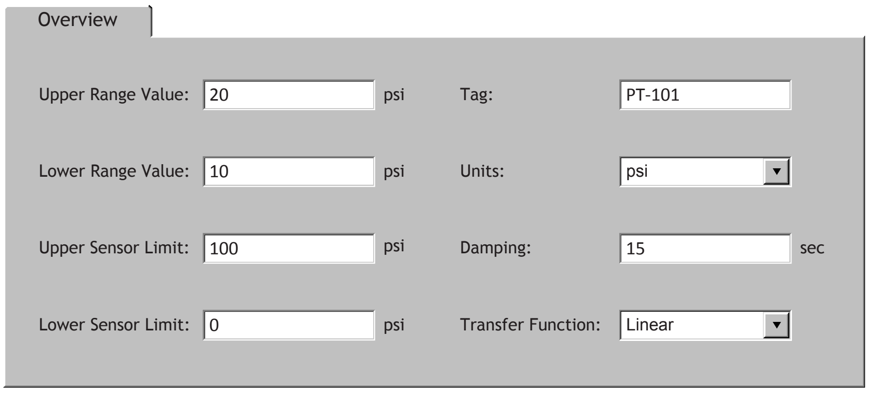
4.5.4.2 Multi-column and -row layout

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

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

Figure 26 – Example of an EDD using COLUMNBREAK

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



Overview

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

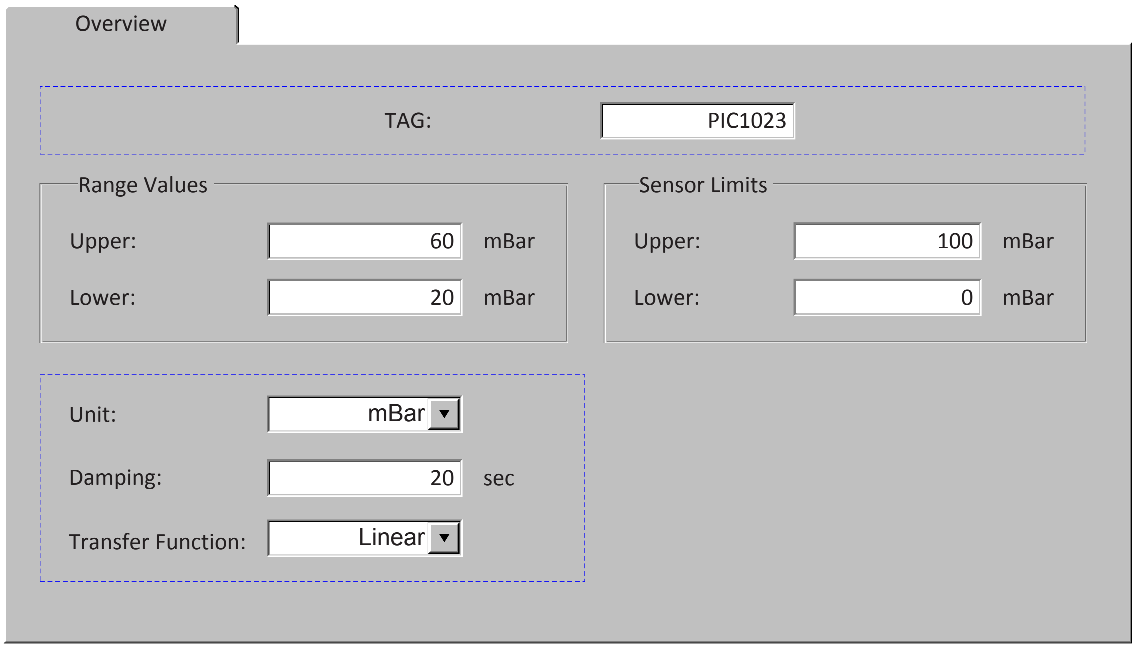
Figure 27 – Example of an EDD application for an overview window

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

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

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


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

Figure 28 – EDD example for an overview window

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



TAG: PIC1023

Range Values
Upper: 60 mBar
Lower: 20 mBar

Sensor Limits
Upper: 100 mBar
Lower: 0 mBar

Unit: mBar
Damping: 20 sec
Transfer Function: Linear

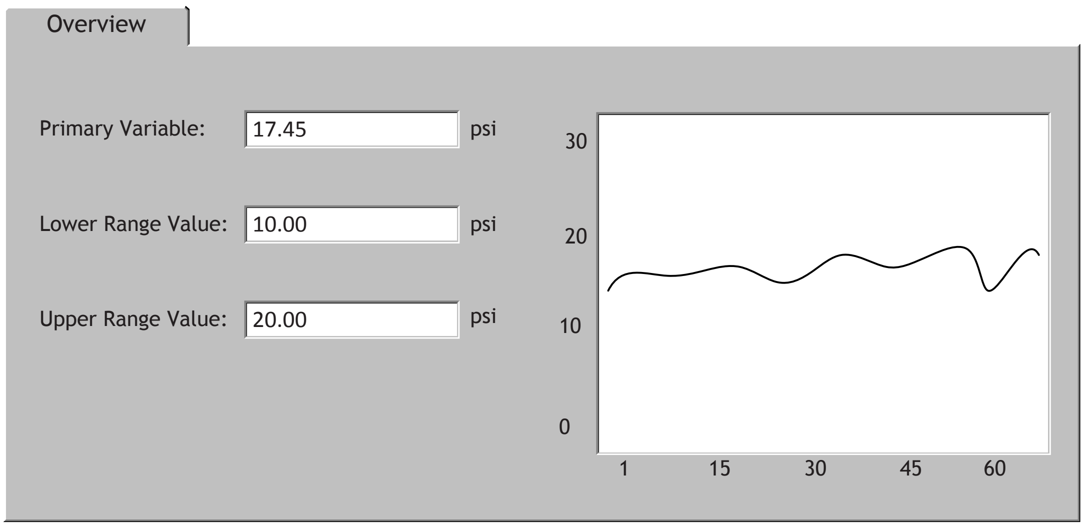
Figure 29 – Example of an EDD application for an overview window

4.5.4.3 In-line graphs and charts

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

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

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



Primary Variable: 17.45 psi

Lower Range Value: 10.00 psi

Upper Range Value: 20.00 psi

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

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

4.5.4.4 Full-width graphs and charts

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

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

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

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

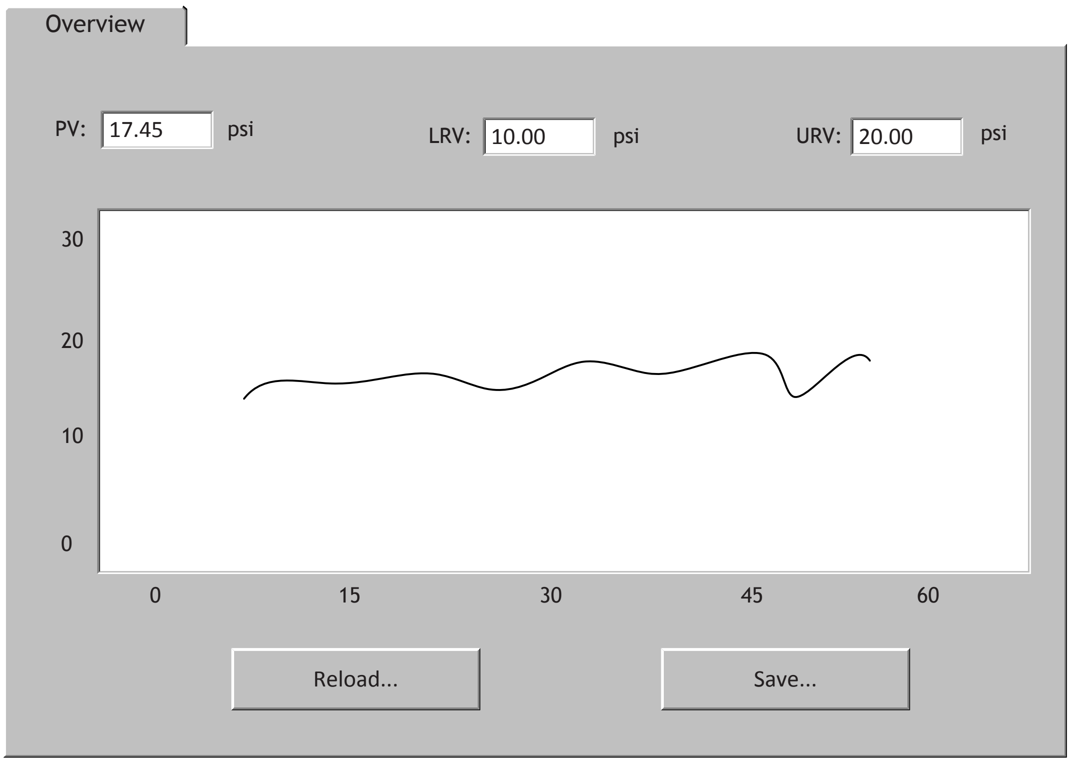


Figure 33 – Example of an EDD application for a full-width graph

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

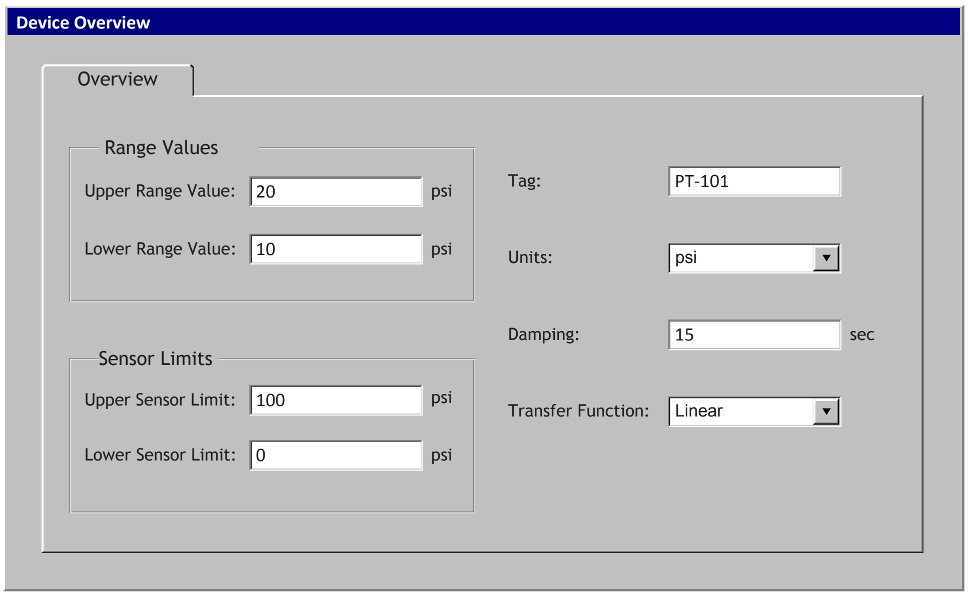
NOTE There are three columns prior to the graph and two columns following the graph.

4.5.4.5 Nested containers

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

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

Figure 34 – Example of an EDD for nested containers


Device Overview

Overview

Range Values

Upper Range Value: 20 psi

Lower Range Value: 10 psi

Sensor Limits

Upper Sensor Limit: 100 psi

Lower Sensor Limit: 0 psi

Tag: PT-101

Units: psi

Damping: 15 sec

Transfer Function: Linear

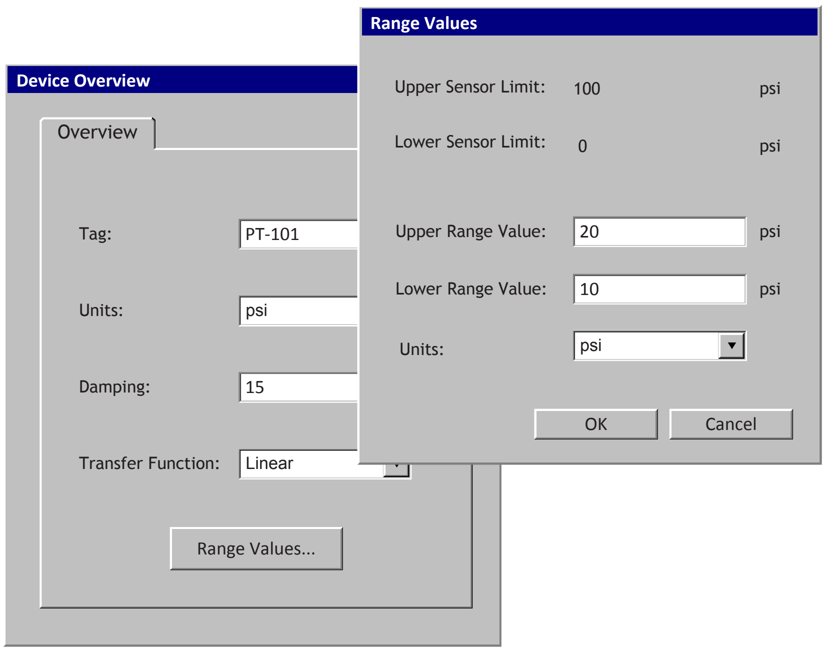
Figure 35 – Example of an EDD application for nested containers

4.5.4.6 Edit displays

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

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

Figure 36 – Example of an EDD for EDIT_DISPLAYS



Tag: PT-101
Units: psi
Damping: 15
Transfer Function: Linear

Upper Sensor Limit: 100 psi
Lower Sensor Limit: 0 psi
Upper Range Value: 20 psi
Lower Range Value: 10 psi
Units: psi

Figure 37 – Example of an EDD application for EDIT_DISPLAYS

4.5.4.7 Images

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

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

Figure 38 – Example of an EDD for images

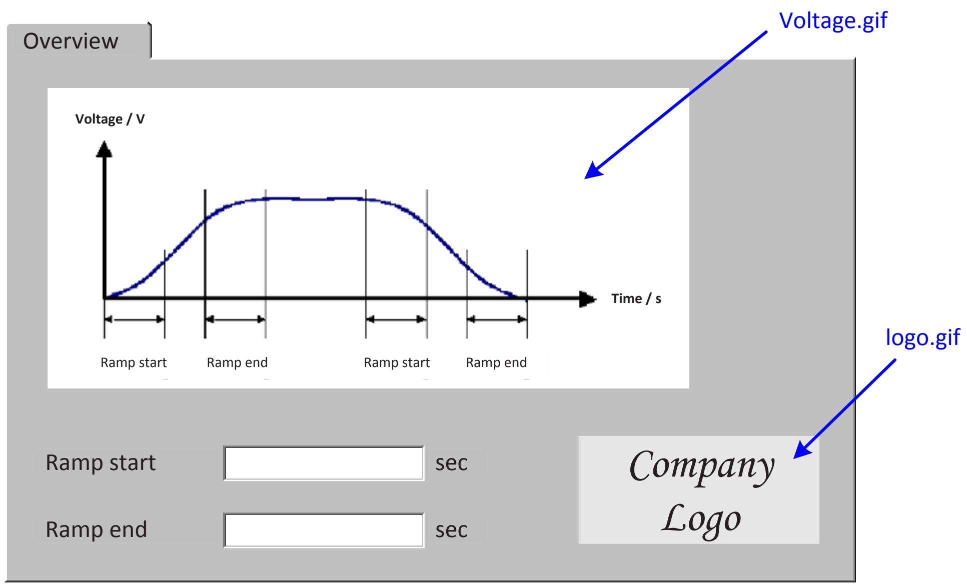


Figure 39 – Example of an EDD application for images

4.5.5 Conditional user interface

4.5.5.1 Overview

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

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

4.5.5.2 VISIBILITY

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

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

4.5.5.3 Conditional expressions in MENU ITEM

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

4.5.5.4 VALIDITY

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

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

NOTE Communication is influenced by VALIDITY, see Clause 10.

4.6 Graphical elements

4.6.1 Overview

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

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

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

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

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

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

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

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

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

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

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

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

4.6.2 Graph and chart

4.6.3 Common attributes

4.6.3.1 Sizes

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

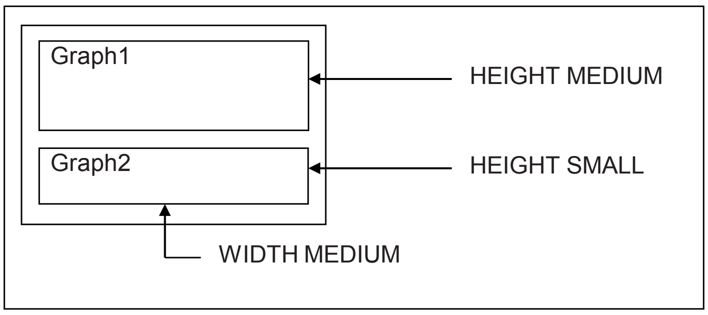


Figure 40 – HEIGHT and WIDTH attributes for CHART and GRAPH

4.6.3.2 Line characteristics

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

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

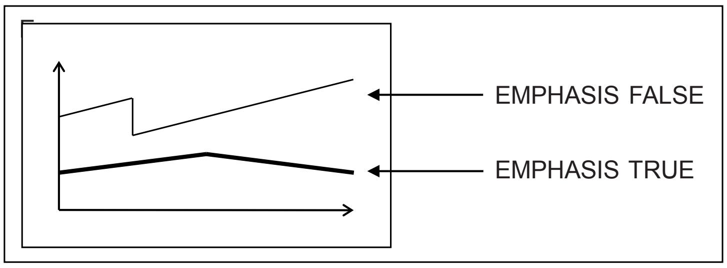


Figure 41 – EMPHASIS attribute to differentiate one or more SOURCEs or WAVEFORMs

4.6.4 CHART

4.6.4.1 Overview

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

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

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

4.6.4.2 Chart types

4.6.4.2.1 Overview

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

4.6.4.2.2 GAUGE

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

4.6.4.2.3 HORIZONTAL_BAR

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

4.6.4.2.4 SCOPE

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

4.6.4.2.5 STRIP

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

4.6.4.2.6 SWEEP

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

4.6.4.2.7 VERTICAL_BAR

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

4.6.4.3 EDDL application without full chart support

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

4.6.4.4 Length and cycle time

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

4.6.4.5 Data sources of a chart

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

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

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

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

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

SOURCE primary_value_source
{
        MEMBERS
        {
                PRIM_VAL, primary_value;
        }
}

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

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

4.6.4.6 CHARTs with multiple SOURCEs referencing the same AXIS

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

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

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

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

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

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

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


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

Figure 43 – Example of a chart with two SOURCEs

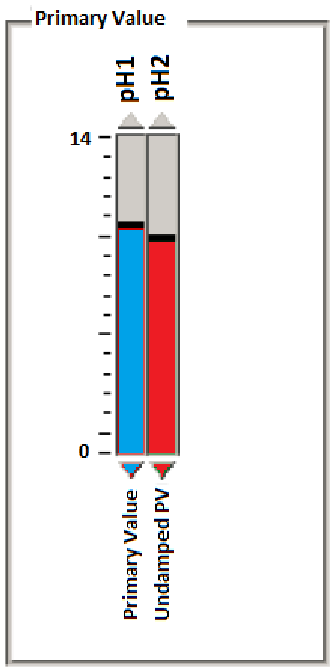


Figure 44 – Displaying example of a chart with two SOURCEs

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

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

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

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

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

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

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

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

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

Figure 45 – Example of a chart with three horizontal bars

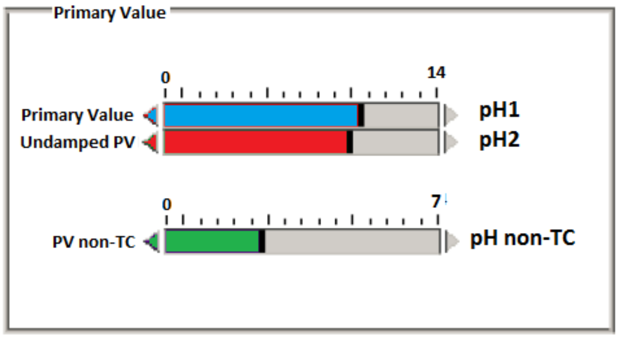


Figure 46 – Displaying example of a chart with three horizontal bars

4.6.4.7 Legend and help for the curves

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

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

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

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

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

4.6.4.8 Zooming and scrolling

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

4.6.4.9 Actions

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

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

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

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

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

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

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

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

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

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

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

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

UNIT
{
        primary_value_unit:
        primary_value,
        calculated_value,
        measuring_values_axis
}

SOURCE calculated_value_source
{
        LABEL "calculated";                     // this label is used in the legend
        LINE_TYPE DATA2;
        Y_AXIS measuring_values_axis;
        MEMBERS
        {
                CALC_VAL, calculated_value;
        }
        INIT_ACTIONS     {CalculationMethod}
        REFRESH_ACTIONS  {CalculationMethod}
}

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

Figure 47 – Example of a chart in a dialog

4.6.5 GRAPH

4.6.5.1 Overview

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

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

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

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

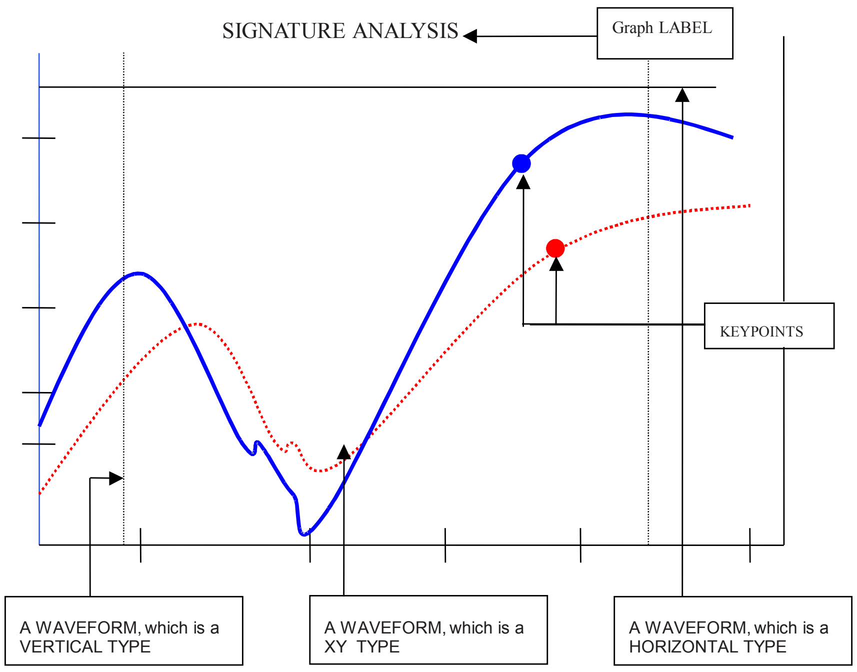

SIGNATURE ANALYSIS
Graph LABEL
KEYPOINTS
A WAVEFORM, which is a VERTICAL TYPE
A WAVEFORM, which is a XY TYPE
A WAVEFORM, which is a HORIZONTAL TYPE

Figure 48 – A graph and the visual elements

4.6.5.2 Types

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

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

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

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

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

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

4.6.5.3 HANDLING

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

4.6.5.4 KEY_POINTS

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

4.6.5.5 Actions

4.6.5.5.1 INIT_ACTIONS

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

4.6.5.5.2 REFRESH_ACTIONS

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

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

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

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

4.6.5.5.3 EXIT_ACTIONS

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

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

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

Figure 49 shows an example of a graph.

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

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

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

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

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

AXIS x_axis_signature
{
       LABEL "x axis";
       MIN_VALUE DEFAULT_X_MIN_VALUE;
       MAX_VALUE DEFAULT_X_MAX_VALUE;
}

AXIS y_axis_signature
{
       LABEL "y axis";
       MIN_VALUE DEFAULT_Y_MIN_VALUE;
       MAX_VALUE DEFAULT_Y_MAX_VALUE;
}

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

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

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

METHOD read_first_signature
{
       DEFINITION
       {
```

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

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

GRAPH valve_signature
{
        MEMBERS
        {
                VAL_SIG, value_signature1;
        }
       X_AXIS x_axis_signature;
}
```

Figure 49 – Example of a graph

4.6.5.6 Axis

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

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

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

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

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

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

GRAPH graph1
{
        MEMBERS
        {
                W1, w1;
                W2, w2;
                W3, w3;
        }
        X_AXIS x1;
}

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

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

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

Figure 50 – Multiple used axes

4.6.5.7 EDDL application without graphical support

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

4.6.5.8 Standard usage

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

4.6.5.9 Integration

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

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

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

4.6.5.10 GRAPH with multiple WAVEFORMs referencing same Y_AXIS

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

4.6.5.11 Multiple waveforms and legends

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

4.6.5.12 Editable graphs

4.6.5.12.1 General

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

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

4.6.5.12.2 POST_EDIT_ACTIONS on variables

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

4.6.5.13 Legend and help of the curves

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

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

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

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

4.6.5.14 Device-supported zooming and scrolling

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

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

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

ARRAY          y_data
{
       NUMBER_OF_ELEMENTS 1000;
       TYPE y_value;
}

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

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

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

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

```
        CLASS LOCAL;
        TYPE TIME;
}

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

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

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

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

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

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

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

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

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

        Y_AXIS          y_axis_signature;

        INIT_ACTIONS { init_signature}
        REFRESH_ACTIONS { refresh_signature}
}

METHOD init_signature
{
        DEFINITION
        {
                // read current data and stored area t_device_min_value,
                // t_ device_max_value,       y_ device_min_value, y_ device_max_value
                read_command (read_current_data);
                // calculation of the waveform data
                value_signature1.X_INITIAL = t_device_max_value;
                value_signature1.X_INCREMENT = (t_device_max_value-t_device_min_value)/
                                              ( y_data.NUMBER_OF_ELEMENTS - 1 );

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

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

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

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

                // reading points and area    t_device_min_value, t_ device_max_value,
                //                            y_ device_min_value, y_ device_max_value from the
device
                read_command (read_data);

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

Figure 51 – EDD with device-supported zooming and scrolling

4.6.6 AXIS

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

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

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

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

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

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

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

4.6.7 IMAGE

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

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

Figure 52 shows an image definition.

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

Figure 52 – EDD example of an IMAGE

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

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

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

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

MENU Self_Test_Menu
{
    …
}
```

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

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

Table 7 – Image formats

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

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

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

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

4.6.8 GRID

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

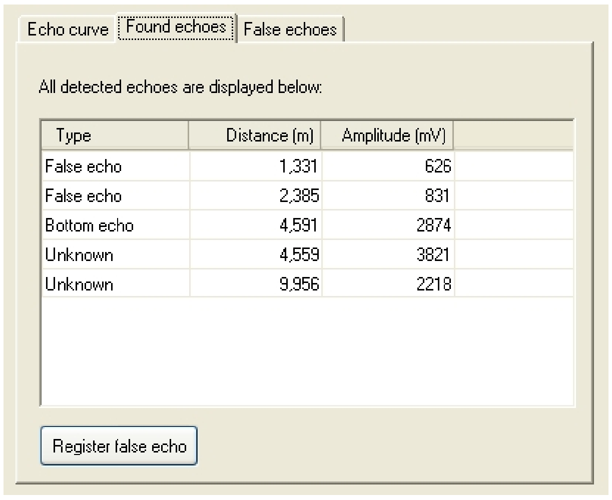
Figure 54 shows a GRID example.

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

ARRAY arrPeakType
{
        NUMBER_OF_ELEMENTS 10;
        TYPE PeakType;
}

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

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

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

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

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

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

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

Figure 54 – EDD example of a GRID

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



All detected echoes are displayed below:

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

Figure 55 – Result of the EDD example

5 EDDL data description

5.1 Variables

5.1.1 VARIABLE TYPEs

5.1.1.1 BIT_ENUMERATED

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

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

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

Figure 56 – Wrong usage of a BIT_ENUMERATED variable

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

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

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

Figure 57 – Usage of ENUMERATED instead of BIT_ENUMERATED

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

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

Table 8 – String handling

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

5.1.2 VARIABLE CLASS

5.1.2.1 Overview

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

5.1.2.2 CLASS OPTIONAL

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

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

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

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

5.1.3 VARIABLE ACTIONS

5.1.3.1 PRE_EDIT_ACTIONS

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

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

5.1.3.2 POST_EDIT_ACTIONS

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

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

5.1.3.3 REFRESH_ACTIONS

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

5.2 EDDL application stored device data

5.2.1 Overview

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

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

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

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

5.2.2 FILE

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

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

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

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

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

```
VARIABLE valve_position
{
TYPE FLOAT;
}

ARRAY valve_stroke
{
TYPE valve_position;
NUMBER_OF_ELEMENTS 1000;
}

VARIABLE valve_signature_comment
{
TYPE ASCII(64);
}

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

Figure 58 – Example of a file declaration

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

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

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

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

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

Figure 59 shows a GRAPH in a MENU.

```
ARRAY current_stroke { TYPE valve_position; NUMBER_OF_ELEMENTS 1000; }

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

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

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

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


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

Figure 59 – Example of comparing valve signatures

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

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

5.2.3 LIST

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

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

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

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

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

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

LIST recorded_signals {  TYPE signal_info; }

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

Figure 60 – Example of more complex file declaration

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

```
stored_signals.SIGNALS[10]
recorded_signals[10]
```

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

```
stored_signals.SIGNALS[10].SIGNAL
echo_signal;
```

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

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

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

GRAPH review_radar_signal
{
        MEMBERS
        {
                PING, one_echo;
        }
}

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

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

                i = 0;

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

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

                        /* display the graph and the annotations. */

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

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

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

Figure 61 – Example of reviewing the stored radar signals

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

COMMAND ping
{
       NUMBER 128;
       OPERATION COMMAND;

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

COMMAND read_ping_data
{
       NUMBER 129;
       OPERATION READ;

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

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

GRAPH new_radar_signal
{
       MEMBERS
       {
              PING, new_echo;
       }
}

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

```
        }
}

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

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

         i = 0;

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

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

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

                 case 0:        /* save the signal */
                     get_dev_var_value (conditions);
                     select_from_list("save where?","front of list; end of list;
                                         insert into list", ans);
                     switch (ans) {
                         case 0: i = 0; break;                     /* begining */
                         case 1: i = recorded_signals.COUNT; break;    /* end */

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

                             /* display the graph and the annotations. */
                                 MenuDisplay (review_radar_signal_menu,
                 “OK”, selector);
                                 acknowledge ( "Radar Signal by %{1} \n" \
                                                   "Description %{2}" varid);

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

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

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

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

                     /* display the graph and the annotations. */

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

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

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

                     /* display the graph and the annotations. */

                         MenuDisplay (review_radard_signal_menu, “OK”, selector);
                         acknowledge ( "Radar Signal by %{1} \
                                             nDescription %{2}" varid);
                         select_from_list ("compare with this signal?",

                                               "Yes;No",ans);

                         if (ans == 0) {
                             MenuDisplay ( compare_radar_signal_menu, “OK”, selector);
                             select_from_list ("compare with another
                                                        signal?","Yes;No",ans);
                             if (ans==0) {
                                 continue;
                             }
                             break;
                         }
                         i++;
                     } while ( i < recorded_signals.COUNT);
                     acknowledge("no signals compared");
                     break;

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

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

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

5.3 Exposing data items outside the EDD application

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

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

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

5.4 Initialization of EDD instances

5.4.1 Overview

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

5.4.2 Initialization support

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

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

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

5.4.3 TEMPLATE

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

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

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

Figure 63 – Example of TEMPLATE usage

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

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

5.5 Device model mapping

5.5.1 BLOCK_A

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

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

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

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

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

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

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

Figure 64 – Example of a BLOCK_A

5.5.2 BLOCK_B

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

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

6 EDDL METHOD programming and usage of Builtins

6.1 Builtin MenuDisplay

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

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

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

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

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

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

NOTE PROFIBUS and PROFINET support nesting methods with UI Builtins.

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

```
IMAGE deviceimage
{
        PATH "device_image.jpg"
}


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

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

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


METHOD setup_wizard
{
        CLASS INPUT;
        LABEL "Device Setup Wizard";
        DEFINITION
        {
                long select=0;
                long step=0;

                do
                {
                        switch (step)
                        {
                        case 0:
                                MenuDisplay(wizard_step0, "NEXT", select);
                                step++;
                                break;
                        case 1:
                                MenuDisplay(wizard_step1, "BACK;NEXT", select);
                                if (select==0) step=0;
                                if (select==1) step=2;
                                break;
                        case 2:
                                MenuDisplay(wizard_step2, "FINISH", select);
                                step++;
                                break;
                        }
                }
                while (step<3);
        }
}
```

Figure 65 – Example of a wizard

6.2 Division by zero and undetermined floating values

6.2.1 Integer and unsigned integer values

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

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

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

6.2.2 Floating-point values

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

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

Table 9 – Examples of floating-point results

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

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

7 Modular devices

7.1 Overview

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

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

7.2 EDD identification

Two identification techniques exists for EDDs.

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

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

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

Table 10 – Usages of COMPONENT_PATH

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

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

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

7.3 Instance object model

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

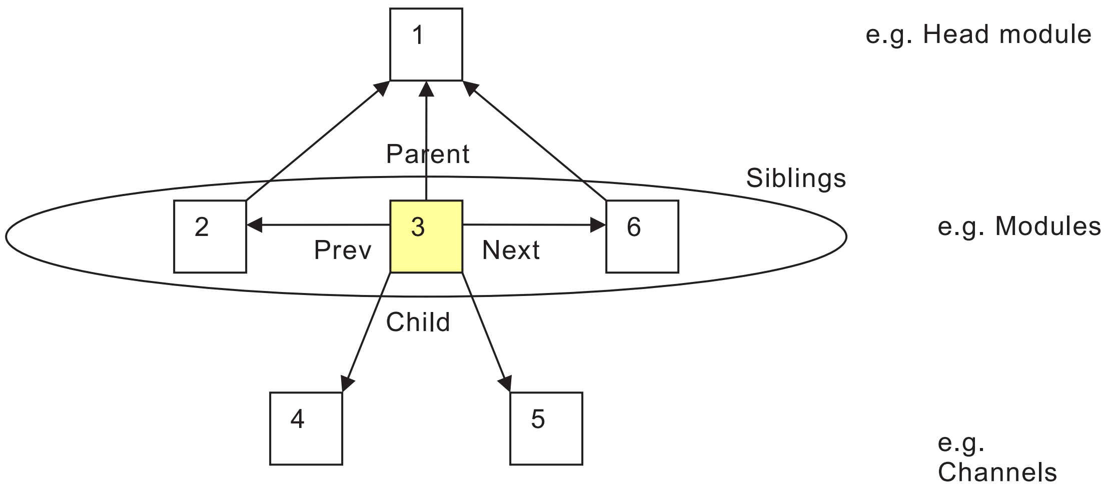


Figure 66 – The different relations of a module

7.4 Offline configuration

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

7.5 Online configuration

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

7.6 Simple modular device example

7.6.1 General

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

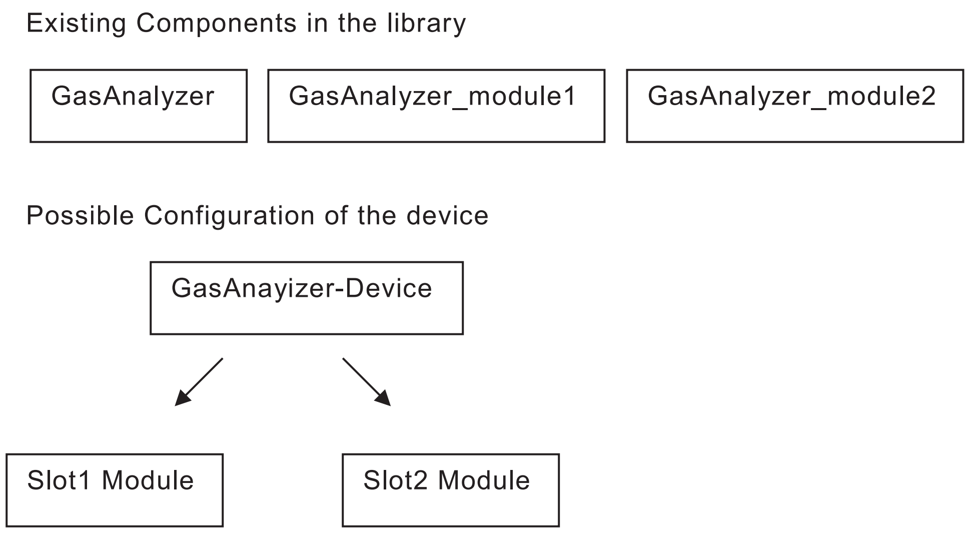


Figure 67 – Components and possible configuration of the modular devices

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

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

7.6.2 Separate EDD file example with direct EDD referencing

Figure 68 shows an EDD example for a modular device.

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

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

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

       COMPONENTS            { GasAnalyzer_module1}

       MINIMUM_NUMBER 0;
       MAXIMUM_NUMBER 2;
}


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

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

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

Figure 68 – Separate EDD file example with direct EDD referencing

Figure 69 shows the EDD example for module1.

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

COMPONENT GasAnalyzer_module1
{
       LABEL                 "GasAnalyzer Module 1";
}

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

Figure 69 – EDD example for module1

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

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

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

COMPONENT_RELATION GasAnalyzer_parent_relation
{
       RELATION_TYPE         PARENT_COMPONENT;
       COMPONENTS            { GasAnalyzer }
}

COMPONENT_REFERENCE GasAnalyzer
{
       DEVICE_TYPE 0x1000
       DEVICE_REVISION 0x01
}

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

Figure 70 – EDD example for module2

7.6.3 Separate EDD file example with classification EDD referencing and interfaces

Figure 71 shows an EDD example for a modular device.

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

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

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

        COMPONENT_RELATIONS     { GasAnalyzer_module_relation }
        SUPPLIED_INTERFACE      { GasAnalyzer_interface }
}

COMPONENT_RELATION GasAnalyzer_module_relation
{
        LABEL                   "GasAnalyzer_module_relation";
        RELATION_TYPE           CHILD_COMPONENT;

        COMPONENTS              { GasAnalyzer_module1}

        REQUIRED_INTERFACE      { module_interface }
        SUPPLIED_INTERFACE      { GasAnalyzer_interface }

        MINIMUM_NUMBER          0;
        MAXIMUM_NUMBER          2;
}

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

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

Figure 71 – EDD example for modular device

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

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

#include "module_interface.dd"

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

        SUPPLIED_INTERFACE      { module_interface }

        INITIAL_VALUES
        {
                module_type     = 1;
        }
}
```

Figure 72 – EDD example for module1

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

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

#include "module_interface.dd"

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

        COMPONENT_RELATIONS     { GasAnalyzer_parent_relation }
        SUPPLIED_INTERFACE      { module_interface }

        INITIAL_VALUES
        {
                module_type     = 2;
        }
}

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

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

Figure 73 – EDD example for module2

7.6.4 One EDD file example

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

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

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

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

COMPONENT GasAnalyzer_module1
{
       LABEL                 "GasAnalyzer Module 1";

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

COMPONENT GasAnalyzer_module2
{
       LABEL                 "GasAnalyzer Module 2";

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

VARIABLE device_mode
{
       LABEL "Mode";
       TYPE ENUMERATED
       {
              {1, "simple"},
              {2, "complex"}
       }
}
MENU   root_menu
{
       LABEL  "Main Menu";
       ITEMS
       {
```

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

Figure 74 – EDD example for module2

7.6.5 Combination of single and separate modular device example

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

7.7 COMPONENT_RELATION

7.7.1 General

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

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

7.7.2 NEXT_COMPONENT usage

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

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

        COMPONENTS              { }
}
```

Figure 75 – NEXT_COMPONENT usage

7.7.3 REQUIRED_RANGES and ADDRESSING usage

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

```
COMPONENT_RELATION GasAnalyzer_module_relation
{
    RELATION_TYPE       CHILD_COMPONENT;
    ADDRESSING          { slot }

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

    MINIMUM_NUMBER      0;
    MAXIMUM_NUMBER      10;
}
```

Figure 76 – REQUIRED_RANGES usage

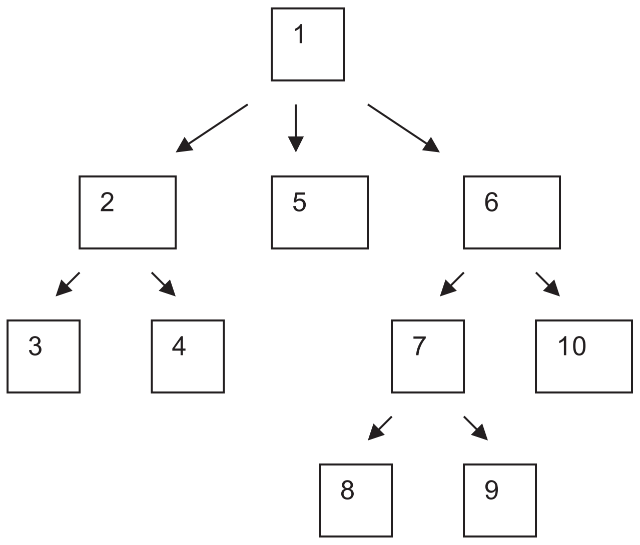
7.8 Upload and download for modular devices

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

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

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

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



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

Figure 77 – Upload/download order of a modular device

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

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

7.9 Diagnostic

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

The diagnostic classification should be extended with two additional cases.

Additional diagnostic classifications are shown in Table 11.

Table 11 – Diagnostic classifications

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

7.10 Reading modular device topology

7.10.1 SCAN

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

Figure 78 shows an example of a device SCAN METHOD.

```
...

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

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

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

LIST MyScanList
{
        ...
        TYPE ComponentInfo;
}

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

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

                        i++;
                }

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

                return FAILURE;
}
}
```

Figure 78 – Example of a SCAN METHOD

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

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

7.10.2 Detect module type

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

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

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

Figure 79 shows an example of a device DETECT METHOD.

```
...

COMPONENT MyDevice
{
        ...
        DETECT          MyDetect;
}

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

Figure 79 – Example of a DETECT METHOD

The following rules apply for SCAN and DETECT.

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

7.11 Configuration check

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

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

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

COMPONENT MyDevice
{
        CHECK_CONFIGURATION           MyConfigCheck;
}

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

Figure 80 – Example of a CHECK_CONFIGURATION METHOD

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

8 Edit session

8.1 Data management

8.1.1 Overview

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

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

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

Figure 81 shows the data caching for an offline session.

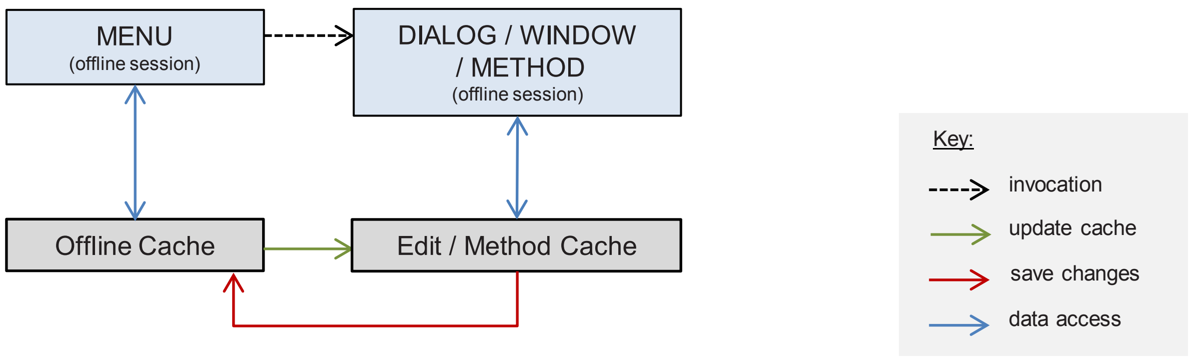


Figure 81 – Data caching for an offline session

For online sessions the EDD application shall use an online cache. The values of the requested CLASS LOCAL variables shall be retrieved from the offline cache for PROFIBUS and PROFINET devices; otherwise the values of the requested CLASS LOCAL variables shall be initialized with DEFAULT_VALUEs. The values of the requested non-LOCAL variables shall be read from the device.

Figure 82 shows the data caching for an online session.

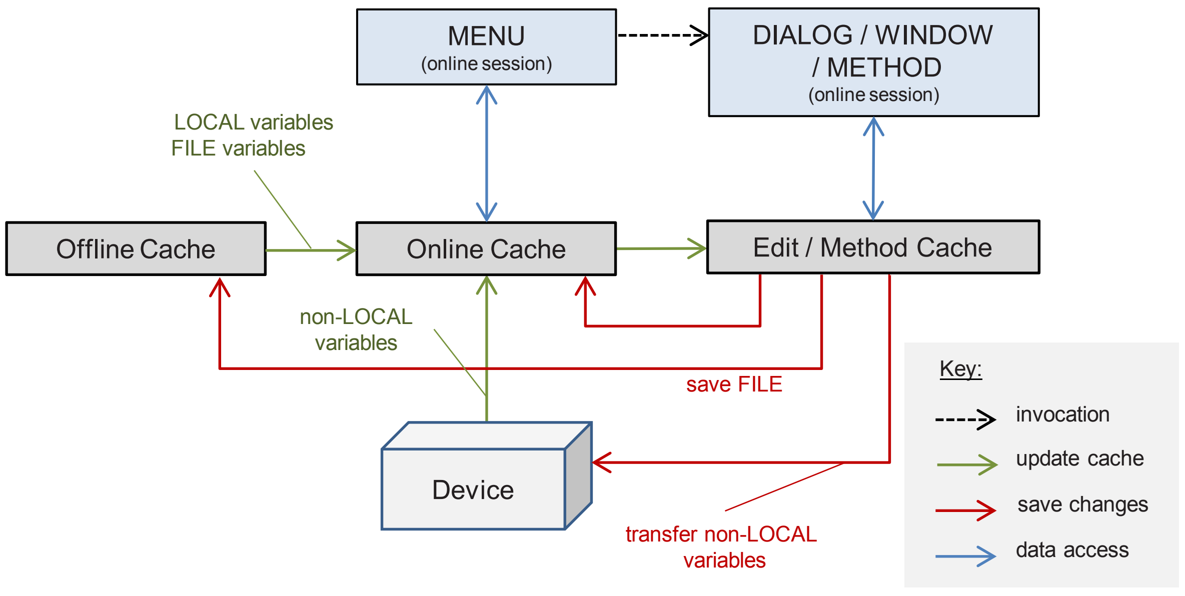


Figure 82 – Data caching for an online session

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

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

8.1.2 General rules

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

Any conditionals shall be evaluated using the current cache.

FOUNDATION fieldbus caches shall support multiple block instances.

8.1.3 Data caching for dialogs and windows

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

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

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

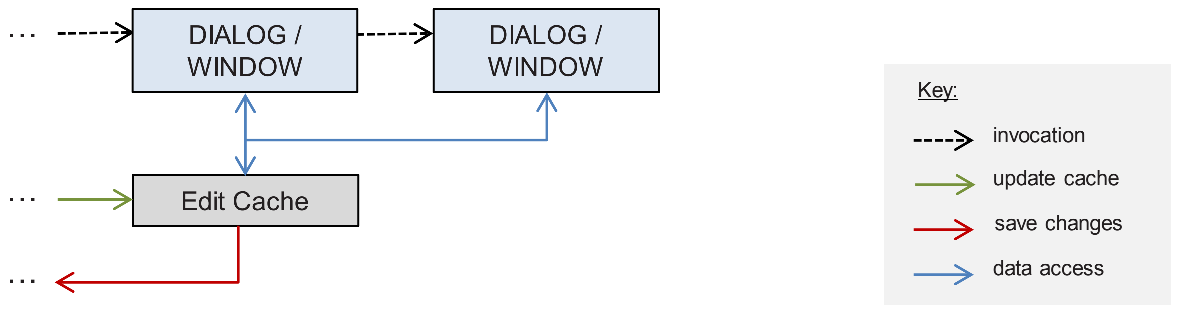


Figure 83 – Sub dialogs or windows using a shared edit cache

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

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

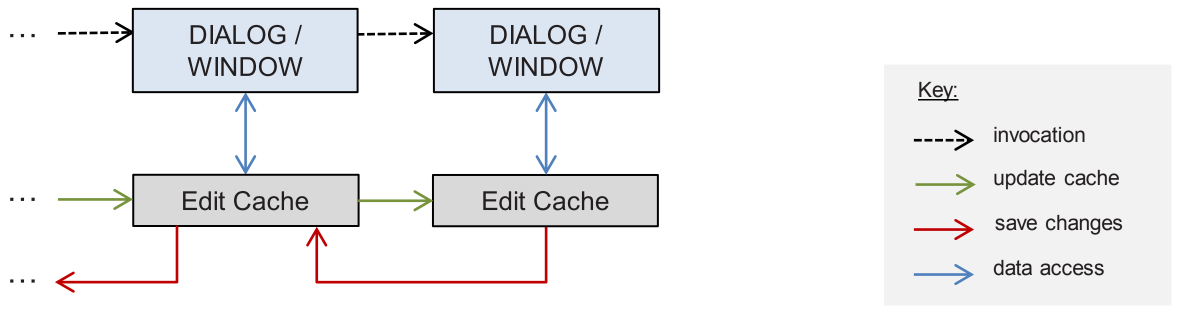


Figure 84 – Sub dialogs or windows using separate edit caches

8.1.4 Data caching for METHODs

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

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

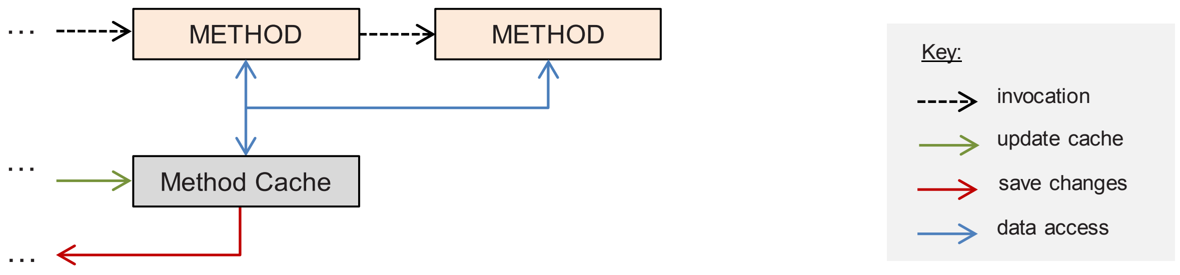


Figure 85 – Data caching for nested METHODs

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

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

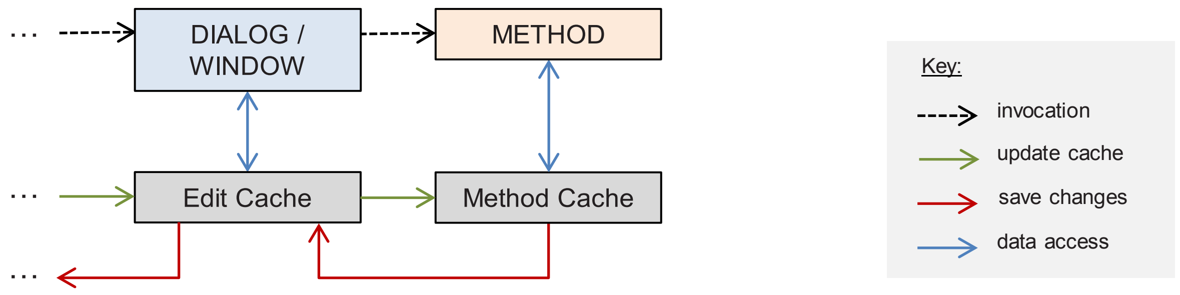


Figure 86 – Data caching for a METHOD invoked within a dialog

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

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

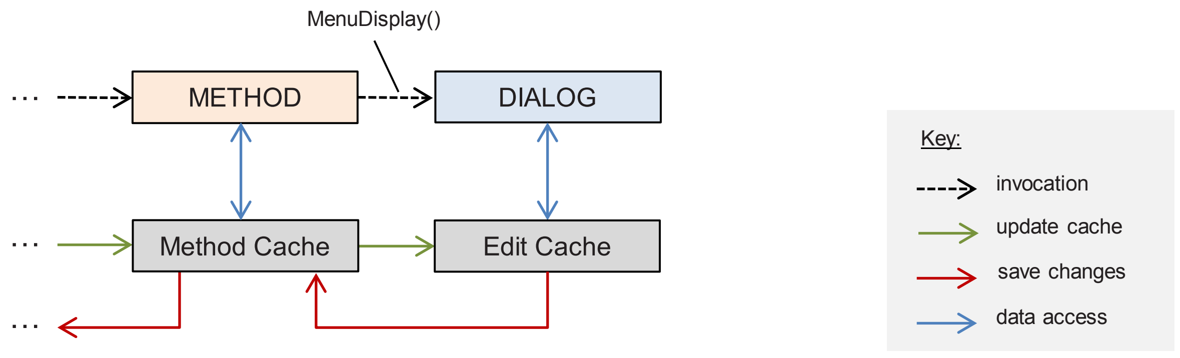


Figure 87 – Data caching for a METHOD invoking a dialog using an edit cache

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

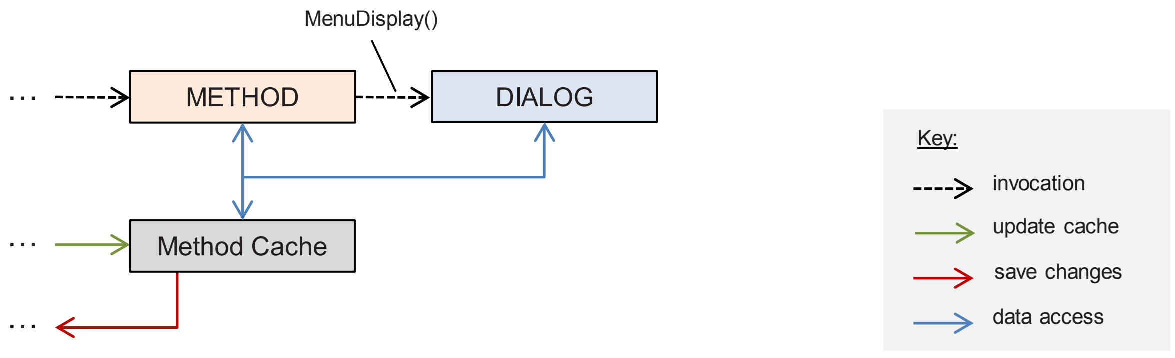


Figure 88 – Data caching for a METHOD invoking a dialog

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

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

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

Table 12 – Builtins for method cache controlling

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

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

8.2 UI aspects of editing sessions

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

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

8.3 User roles

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

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

9 Offline and online configuration

9.1 Overview

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

Device parameters can be differentiated into application-specific parameters and device-specific parameters. To use a new device at a plant location, the application-specific parameter should be downloaded into the device. When a device is replaced, the application-specific parameters from the old device should be downloaded into the new device. However, the device-specific parameters are never downloaded because they are specific to the physical device.

9.2 Offline dataset

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

9.3 Offline configuration

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

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

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

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

9.4 Online dataset

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

9.5 Online configuration

The online configuration is specified by:

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

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

9.6 Upload and download

9.6.1 Overview

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

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

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

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

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

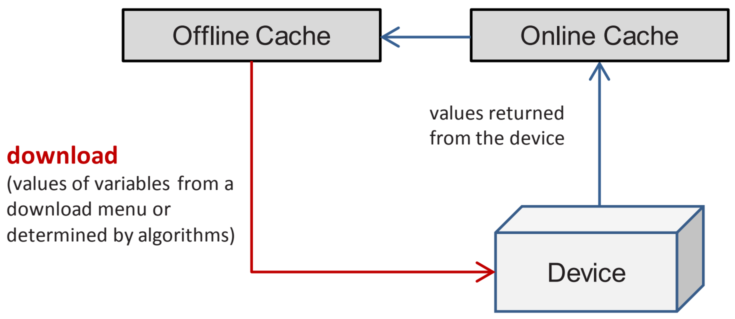

Offline Cache
Online Cache
values returned from the device
download
(values of variables from a download menu or determined by algorithms)
Device

Figure 89 – Data flow for download to the device

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

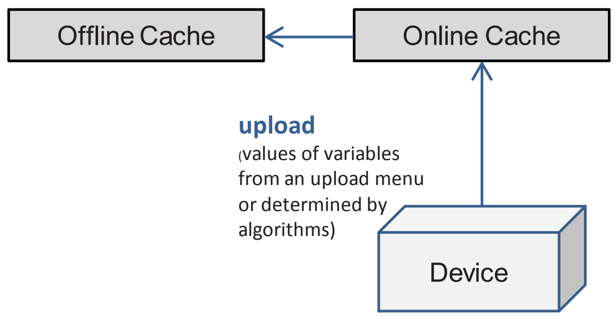

Offline Cache
Online Cache
upload
(values of variables from an upload menu or determined by algorithms)
Device

Figure 90 – Data flow for upload from the device

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

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

9.6.2 Error recovery

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

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

9.6.3 Upload procedure

9.6.3.1 Overview

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

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

Table 13 – List of defined upload menu identifiers

Menu identifiers
upload_from_device_root_menu
download_variables

9.6.3.2 General rules

The following are the general rules for the upload procedure.

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

9.6.3.3 Upload with upload menu

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

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

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

9.6.3.4 Upload without upload menu

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

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

   For BLOCK_A devices the EDD application shall read items in the PARAMETERS attribute.
2) The order of the items to be read shall be determined according to VARIABLEs needed by conditionals, UNIT and REFRESH relations. The cause-parameters of UNIT and REFRESH relations should be read before the effected-parameters. The ordering of items in unit and refresh relations cause or effect lists are not used for ordering the communication.
   Any data item not ordered by the rules in 9.6.3 can be read in any order. Devices shall not assume a specific order.

9.6.3.5 Using online dataset for upload

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

9.6.4 Download procedure

9.6.4.1 Overview

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

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

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

Table 14 – List of defined download menu identifiers

Menu identifiers
download_to_device_root_menu
upload_variables

9.6.4.2 General rules

The following are the general rules for the download procedure.

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

9.6.4.3 Download with download menu

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

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

9.6.4.4 Download without download menu

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

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

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

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

The following rules apply to FOUNDATION fieldbus devices.

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

10 EDDL communication description

10.1 COMMAND

10.1.1 General

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

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

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

10.1.2 OPERATION

10.1.2.1 General

This attribute specifies the purpose of the command.

10.1.2.2 OPERATION READ

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

10.1.2.3 OPERATION WRITE

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

10.1.2.4 OPERATION COMMAND

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

10.1.2.5 PROFIBUS and PROFINET communication mapping

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

Table 15 – PROFIBUS and PROFINET communication mapping

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

10.1.3 TRANSACTION

10.1.3.1 General

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

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

10.1.3.2 REPLY and REQUEST

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

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

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

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

10.1.3.3 Data item masks

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

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

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

10.1.3.4 Example of usage of a single item mask

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

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

Figure 91 – Example of a single item mask

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

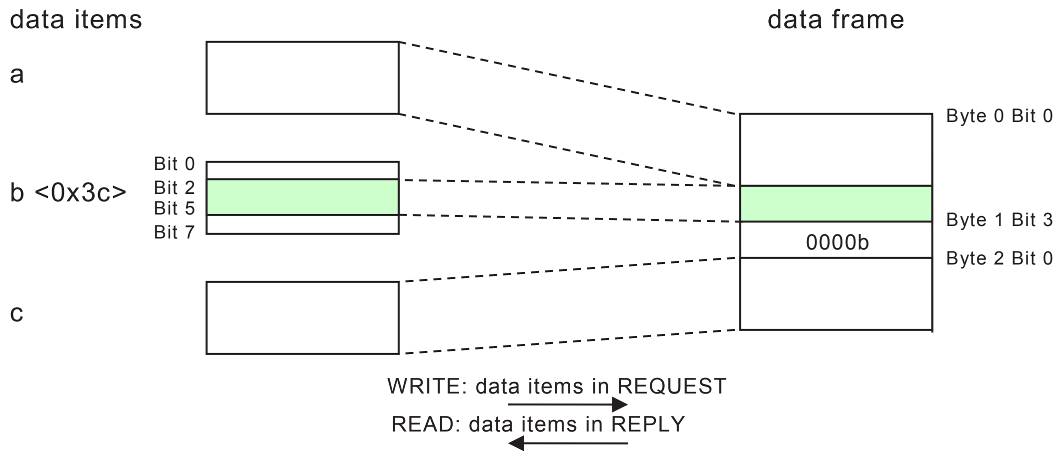


Figure 92 – Mapping example with a single item mask

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

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

Figure 93 – Multiple item masks

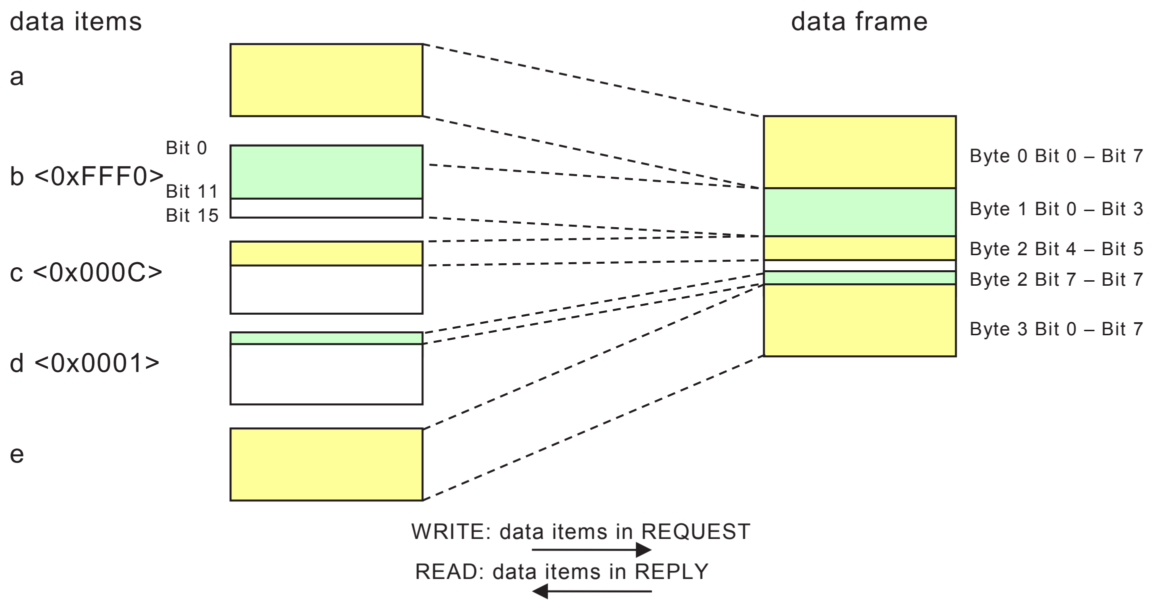


Figure 94 – Mapping example with a multiple item mask

10.1.3.5 Data item qualifiers

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

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

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

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

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

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

Figure 95 – INFO qualifier

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

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

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

Figure 96 – INDEX qualifier

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

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

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

Figure 97 – INFO and INDEX qualifier

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

10.1.3.6 RESPONSE_CODE

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

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

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

10.1.4 Command addressing

10.1.4.1 HART Command addressing

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

10.1.4.2 PROFIBUS DP addressing

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

10.1.4.3 PI Profile for Process Control Devices Addressing

The absolute addresses of BLOCKs of a PI Profile for Process Control Devices device are stored in the device management directory. The EDD application has to fetch the directory and to resolve the base address of the BLOCK (BLOCK_B). The BLOCK attribute of a COMMAND is a reference to a BLOCK_B item.

A BLOCK_B EDDL item is described with a TYPE and a NUMBER attribute. The TYPE attribute defines the types of a block (PHYSICAL, TRANSDUCER or FUNCTION). The NUMBER attribute defines the number of a block of a block type in the device management directory. NUMBER starts for all block types with 1.

The command attribute INDEX is a relative index to the first dataset of the block. To address a dataset in the device, the absolute starting slot and starting index from the device management directory are used and the relative INDEX of the COMMAND is added.

10.1.4.4 PROFINET addressing

API, SLOT, SUB_SLOT and INDEX are used to address datasets in the device in an absolute way.

10.1.4.5 Addressing for other protocols

HEADER is a string attribute that allows to contain protocol specific information for a COMMAND. The HEADER attribute is not used for PROFIBUS, PROFINET, FOUNDATION fieldbus or HART devices. The concrete contents of the string is out of the scope of this standard.

10.2 Parsing data received from the device

10.2.1 General

The EDD application shall transfer data received from the device into the requested data items. If the device returns exactly the same amount of data as requested and doesn’t return an error or warning (see also RESPONSE_CODE), the EDD application shall not log an error.

The EDD application shall not change data items which are not received from the device. EDDL application shall not invoke POST_READ_ACTIONS for data items not received.

10.2.2 Parsing complex data items

When a LIST, VALUE_ARRAY, REFERENCE_ARRAY, COLLECTION and RECORD is referenced by itself in a communication (e.g. COMMAND), the EDD application shall fill up data from the device into referenced data item with the following rules:

a) LIST, VALUE_ARRAY and REFERENCE_ARRAY
   shall be filled up starting at the lowest index value
b) COLLECTION and RECORD
   shall be filled up from the first MEMBER

10.2.3 FOUNDATION Fieldbus

The EDD application shall log an error if the device returns less data than specified by RECORDs, VALUE_ARRAYs, PARAMETER_LISTs or VARIABLEs.

If the device returns more data than a PARAMETER_LIST specifies, the EDD application shall ignore additional data without logging an error or warning. For all other data items the EDD application shall log an error if more data is received.

10.2.4 HART

The EDD application shall log a fatal error if the device returns less data than specified by the COMMAND.

If the device returns more data than the COMMAND specifies, the EDD application shall ignore additional data without logging an error or warning.

EDDL application shall not invoke POST_WRITE_ACTIONS for data items not received after a write COMMAND.

10.2.5 PROFIBUS and PROFINET

The EDD application shall log an error if the device returns less data than specified by the COMMAND REPLY unless the last data item is a variable of any string data type. The EDD application shall transfer all received data into this variable.

If the device returns more data than the COMMAND REQUEST specifies, the EDD application shall ignore additional data without logging an error or warning unless the last data item is a LIST. If the last data item is a LIST, the EDD application shall use or create LIST elements until all data from the device are consumed.

10.3 FOUNDATION Fieldbus communication model

FOUNDATION fieldbus devices are block object oriented devices according to IEC 61804-2. Each block is composed of a characteristics and one or more block parameters. In EDDL, each block type is defined using BLOCK_A construct. The BLOCK_A attribute CHARACTERISTICS maps to the block characteristics found in the device. Block parameters can be VARIABLEs, RECORDs and VALUE_ARRAYs.

One or more blocks are represented in an object dictionary. The object dictionary is access by an index number using read and write services. The object dictionary provides a block directory to identify the number and location of all blocks within the object dictionary.

All data items accessible by an EDD application are accessed in the context of a block. The EDD does not provide any information about the object dictionary index of blocks. Instead, the block characteristics provides a reference to the EDD item using a unique symbol id.

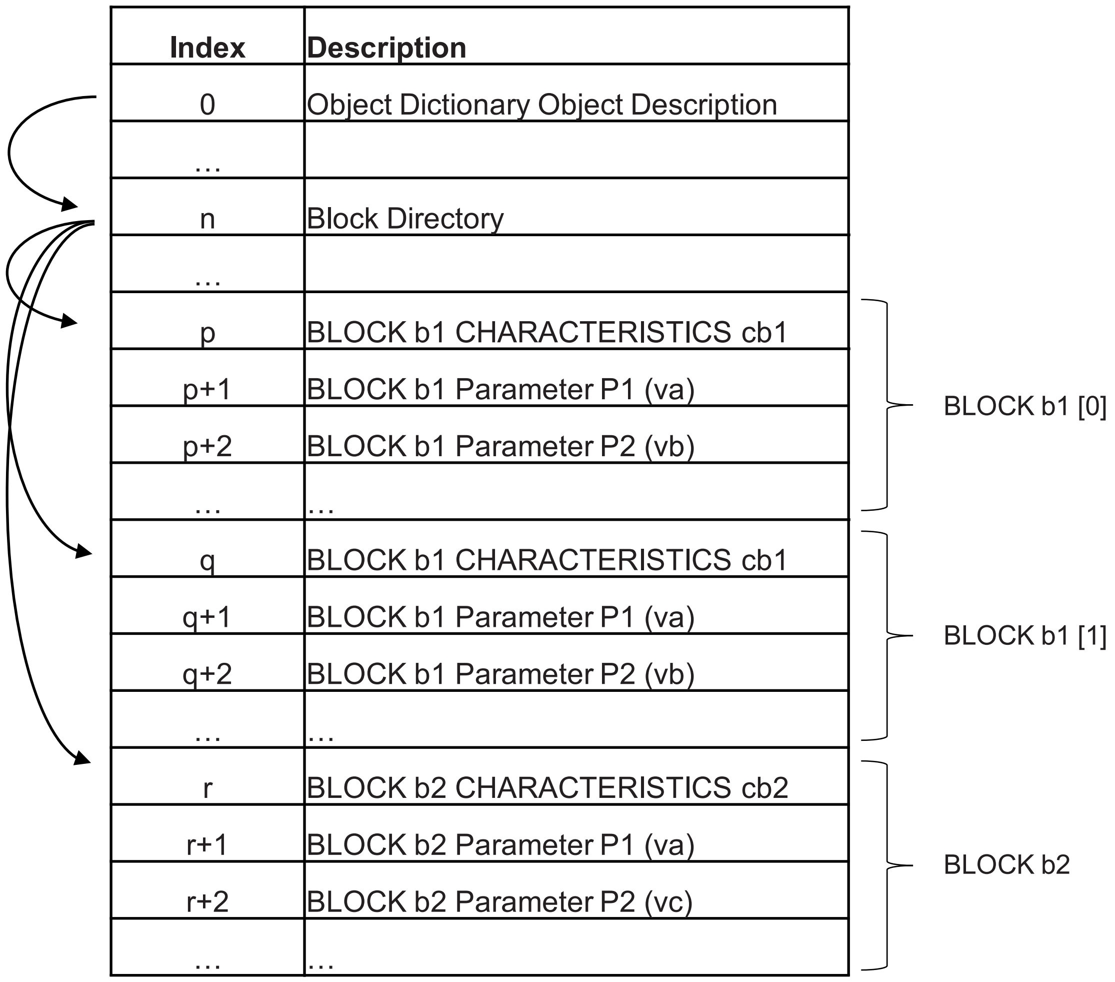
When an EDD is created, unique symbol ids are generated for each EDD item. This symbol id is also encoded in the block characteristics of each block in the device.

By accessing the characteristics of each block to find the associated symbol id, the EDD application associates an EDD BLOCK_A declaration with the block in the device. All BLOCK_A PARAMETERS are listed consecutively after the block CHARACTERISTICS.

By accessing the object dictionary description and block directory, the EDD application can calculate the absolute index to access each BLOCK_A CHARACTERISTICS and PARAMETERS.

A device may have several instances of blocks that will map to a single BLOCK_A declaration in the EDD.

Figure 98 illustrates 2 instances of BLOCK b1 and 1 instance of BLOCK b2 in an object dictionary. The Figure 99 EDDL fragment can be used to describe this example.



Index	Description
0	Object Dictionary Object Description
…	
n	Block Directory
…	
p	BLOCK b1 CHARACTERISTICS cb1
p+1	BLOCK b1 Parameter P1 (va)
p+2	BLOCK b1 Parameter P2 (vb)
…	…
q	BLOCK b1 CHARACTERISTICS cb1
q+1	BLOCK b1 Parameter P1 (va)
q+2	BLOCK b1 Parameter P2 (vb)
…	…
r	BLOCK b2 CHARACTERISTICS cb2
r+1	BLOCK b2 Parameter P1 (va)
r+2	BLOCK b2 Parameter P2 (vc)
…	…

Figure 98 – Example device with 2 unique BLOCK_A definitions

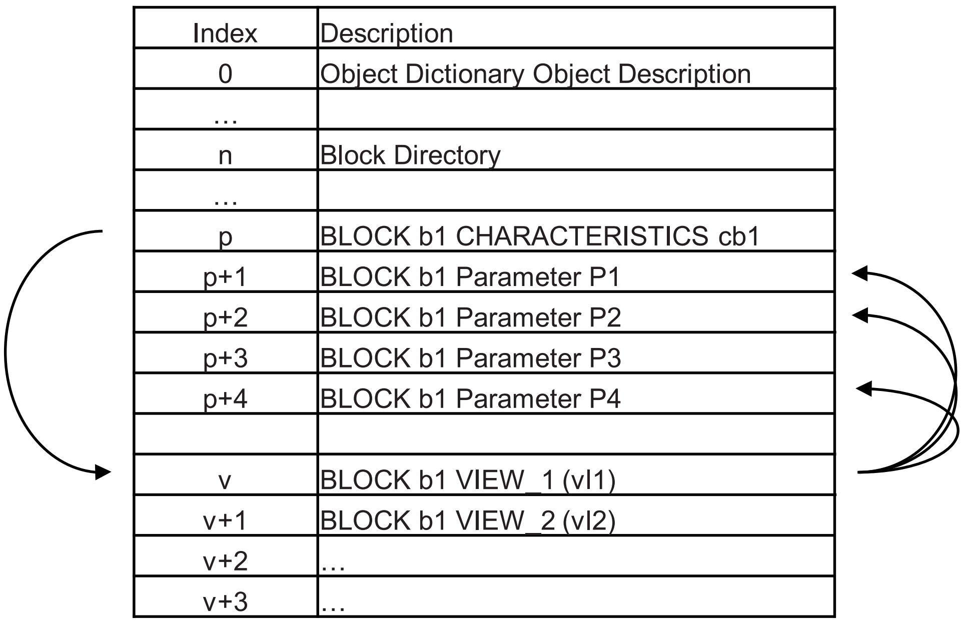
```
BLOCK b1
{
    CHARACTERISTICS cb1;
    PARAMETERS
    {
         P1, va;
         P2, vb;
         /* ... */
     }
}

BLOCK b2
{
    CHARACTERISTICS cb2;
    PARAMETERS
    {
         P1, va;
         P2, vc;
         /* ... */
     }
}

VARIABLE va { /* ... */ }
VARIABLE vb { /* ... */ }
VARIABLE vc { /* ... */ }

RECORD cb1
{
      MEMBERS
      {
          /* ... */
          DD_ITEM, __dd_item;   /* symbol id of block b1 */
          /* ... */
      }
}

RECORD cb2 { /* ... */ }
```

Figure 99 – Example EDD for a device with 2 unique BLOCK_A definitions

Each block also references 4 block views. Block views provides read and write access to several block parameters at a single object dictionary index. Block views provide an efficient means to access multiple parameters from a device in a single communication transaction.

Each view is described in EDDL using the VARIABLE_LIST construct. Each VARIABLE_LIST is associated with a BLOCK_A using the PARAMETER_LIST attribute. The BLOCK CHARACTERISTICS provides mapping to the absolute index of the view. Each view is consecutively listed in the object dictionary

Figure 100 illustrates an example of VARIABLE_LIST vl1 for BLOCK b1. The Figure 101 EDDL fragment can be used to describe this example.



Index	Description
0	Object Dictionary Object Description
…	
n	Block Directory
…	
p	BLOCK b1 CHARACTERISTICS cb1
p+1	BLOCK b1 Parameter P1
p+2	BLOCK b1 Parameter P2
p+3	BLOCK b1 Parameter P3
p+4	BLOCK b1 Parameter P4
v	BLOCK b1 VIEW_1 (vl1)
v+1	BLOCK b1 VIEW_2 (vl2)
v+2	…
v+3	…

Figure 100 – BLOCK_A example with PARAMETER_LISTS

```
BLOCK b1
{
    CHARACTERISTICS cb1;
    PARAMETERS
    {
         P1, va;
         P2, vb;
         P3, ra;
         P4, aa;
    }

    PARAMETER_LISTS
    {
          VIEW_1, vl1;
          VIEW_2, vl2;
           /* … */
    }
}

VARIABLE_LIST vl1
{
       MEMBERS
       {
            VL1, PARAM.P1;
            VL2, PARAM.P2;
            VL2, PARAM.P4;

        }
}

VARIABLE_LIST vl2 { /* ... */ }

VARIABLE va { /* ... */ }
VARIABLE vb { /* ... */ }
RECORD ra   { /* ... */ }
ARRAY aa    { /* ... */ }

RECORD cb1
{
      MEMBERS
      {
          /* ... */
          VIEWS_INDEX, __views_index;   /* index of first view */
          /* ... */
      }
}
```

Figure 101 – Example EDD for a BLOCK_A with PARAMETER_LISTS

11 EDD development

11.1 Dictionaries

Using the dictionary file is convenient to separate localized text from other EDD content. Dictionaries containing stings e.g. of LABEL- or HELP-attributes, prompt strings of Builtin parameters.

The dictionary file can be updated separately from the individual EDD of each device.

If common dictionary entries exist, the EDD developer should choose them instead of defining individual strings in the device specific dictionary or in each device EDD.

Device EDD dictionaries do not overwrite common dictionary definitions. The EDD application shall use a common dictionary definition even if the EDD dictionary has the same entry.

11.2 Reserved

Reserved for other EDD development specifications.

Annex A
(normative)

Device simulation

Device simulation is an optional feature that is only for the EDD development to check EDD behavior. For this purpose some additional entry points will be used by a specific EDD testing tool. This tool runs the EDD in a field device simulator.

The device_simulation_method shall be an identifier of a METHOD that shall only be used by field device simulators. In device simulators, this method shall be run at the monotonic period specified by the device_simulation_background_period VARIABLE.

The device_simulation_background_period shall be an identifier of a VARIABLE that shall only be used by field device simulators for defining the execution period for the device_simulation_method. This VARIABLE shall be of CLASS LOCAL and TYPE FLOAT and its value should be set using the DEFAULT_VALUE attribute. Simulators shall execute background methods with a periodic accuracy of ± 10 ms. device_simulation_background_period shall default to 1,0 s if the VARABLE does not exist or does not have a DEFAULT_VALUE.

Annex B
(informative)

Predefined identifiers

Table B.1 provides predefined identifiers.

Table B.1 – Predefined identifiers

Item type	Identifier	Profile	Description
VARIABLE	comm_status	HART	When bits are set in this VARIABLE there has been a communication error
IMAGE	device_icon	HART	An IMAGE for a device
MENU	device_root_menu	All	See 4.3
MENU	device_root_menu*	FOUNDATION fieldbus	BLOCK_A based MENU optimized for PC based applications
VARIABLE	device_simulation_background_period	HART	See Annex A
METHOD	device_simulation_method	HART	See Annex A
VARIABLE	device_status	HART	A bit set in this VARIABLE indicates a potential problem in the device
MENU	diagnostics_root_menu	All	See 4.3
MENU	diagnostics_root_menu*	FOUNDATION fieldbus	BLOCK_A based MENU optimized for PC based applications
MENU	download_to_device_root_menu	All	See 9.6.4
MENU	download_variables	PROFIBUS, PROFINET	See 9.6.3
VARIABLE	extended_device_status	HART	A bit set in this VARIABLE indicates a potential problem in the device
ARRAY OF VARIABLE	factory_protection_array	HART	This lists VARIABLEs that are not copied from one device instance into new instances.
VARIABLE	hart_functions	HART	Indicates EDD application special features
MENU	hh_device_root_menu	FOUNDATION fieldbus	Device level MENU optimized for handheld applications
MENU	hh_diagnostics_root_menu	FOUNDATION fieldbus	Device level MENU optimized for handheld applications
MENU	hh_process_variables_root_menu	FOUNDATION fieldbus	Device level MENU optimized for handheld applications
MENU	hot_key	HART	Short cut for handheld applications
ARRAY	loop_warning_variables	HART	Array of VARIABLEs that could disrupt, or change the output loop current when they are sent to and stored in the device
MENU	maintenance_root_menu	All	See 4.3
MENU	*Menu_Top*	FOUNDATION fieldbus	BLOCK_A MENU optimized for handheld applications
MENU	Menu_Main_ Maintenance[a]	PROFIBUS	MENU added to the menu bar of the EDD application
MENU	Menu_Main_ Sprecialist[a]	PROFIBUS	MENU added to the menu bar of the EDD application
COLLECTION	no_download*	FOUNDATION fieldbus	Writable COLLECTION of BLOCK_A PARAMETERS that should not be part of a bulk download (e.g. parameters that require interaction with a device for calibration)
MENU	offline_root_menu	All	See 4.3
MENU	OnlineWindow_display[a]	PROFIBUS	MENU containing process variables

Item type	Identifier	Profile	Description
METHOD	post_send_configuration_metho d	HART	METHOD called after download
METHOD	pre_send_configuration_method	HART	METHOD called before download
MENU	process_variables_root_menu*	FOUNDATI ON fieldbus	BLOCK_A based MENU optimized for PC based applications
MENU	process_variables_root_menu[a]	All	See 4.3
COMMAND	read_additional_device_status	HART	COMMAND to read additional diagnostic information
VARIABLE	response_code	HART	This VARIABLE contains the response code for the command response received
MENU	root_menu	HART	See 4.2
VARIABLE	std_ResponseCode	PROFIBUS, PROFINET	Standard communication response for PROFIBUS and PROFINET
MENU	Table_Main_ Maintenance[a]	PROFIBUS	A starting point for the explorer view of a device
MENU	Table_Main_ Sprecialist[a]	PROFIBUS	A starting point for the explorer view of a device
MENU	upload_from_device_root_menu	All	See 9.6.3
MENU	upload_variables	HART, PROFIBUS, PROFINET	See 9.6.4
COLLECTION	upload_wanted*	FOUNDATI ON fieldbus	Read-only COLLECTION of BLOCK_A PARAMETERS that should be included in an offline dataset

Key:

All: HART, FOUNDATION fieldbus, PROFIBUS, PROFINET

*: prefix and postfix for identifiers

[a] These identifiers should not be used to write new EDDs.

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.

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 Group Headquarters

389 Chiswick High Road London W4 4AL UK