IEC 61804-4

Edition 2.0 2020-06

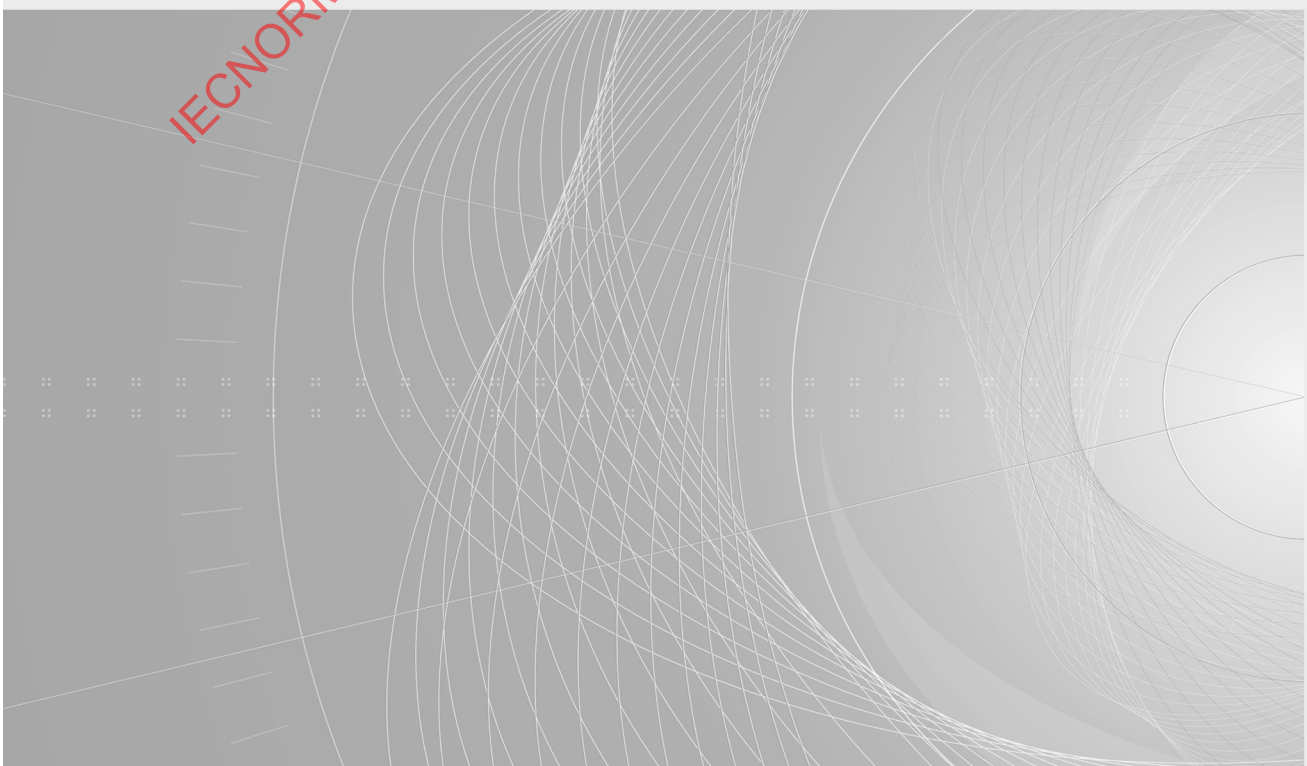
INTERNATIONAL STANDARD

NORME INTERNATIONALE

colour inside

Devices and integration in enterprise systems – Function blocks (FB) for process control and electronic device description language (EDDL) – Part 4: EDD interpretation

Les dispositifs et leur intégration dans les systèmes de l'entreprise – Blocs fonctionnels (FB) pour les procédés industriels et le langage de description électronique de produit (EDDL) – Partie 4: Interprétation EDD



IEC 61804-4:2020-06(en-fr)

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

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

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

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

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 It also gives information on projects, replaced and withdrawn publications.

IEC Just Published - webstore.iec.ch/justpublished
Stay up to date on all new IEC publications. Just Published details all new publications released. Available online and once a month by email.

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

Electropedia - www.electropedia.org
The world's leading online dictionary on electrotechnology, containing more than 22 000 terminological entries in English and French, with equivalent terms in 16 additional languages. Also known as the International Electrotechnical Vocabulary (IEV) online.

IEC Glossary - std.iec.ch/glossary
67 000 electrotechnical terminology entries in English and French extracted from the Terms and Definitions clause of IEC publications issued since 2002. Some entries have been collected from earlier publications of IEC TC 37, 77, 86 and CISPR.

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

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

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

IEC Just Published - webstore.iec.ch/justpublished
Restez informé sur les nouvelles publications IEC. Just Published détaille les nouvelles publications parues. Disponible en ligne et une fois par mois par email.

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

Electropedia - www.electropedia.org
Le premier dictionnaire d'électrotechnologie en ligne au monde, avec plus de 22 000 articles terminologiques en anglais et en français, ainsi que les termes équivalents dans 16 langues additionnelles. Egalement appelé Vocabulaire Electrotechnique International (IEV) en ligne.

Glossaire IEC - std.iec.ch/glossary
67 000 entrées terminologiques électrotechniques, en anglais et en français, extraites des articles Termes et Définitions des publications IEC parues depuis 2002. Plus certaines entrées antérieures extraites des publications des CE 37, 77, 86 et CISPR de l'IEC.

IEC 61804-4

Edition 2.0 2020-06

INTERNATIONAL STANDARD

NORME INTERNATIONALE

colour inside

Devices and integration in enterprise systems – Function blocks (FB) for process control and electronic device description language (EDDL) – Part 4: EDD interpretation

Les dispositifs et leur intégration dans les systèmes de l'entreprise – Blocs fonctionnels (FB) pour les procédés industriels et le langage de description électronique de produit (EDDL) – Partie 4: Interprétation EDD

INTERNATIONAL ELECTROTECHNICAL COMMISSION

COMMISSION ELECTROTECHNIQUE INTERNATIONALE

ICS 25.040.40; 35.240.50

ISBN 978-2-8322-8445-2

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

_____

DEVICES AND INTEGRATION IN ENTERPRISE SYSTEMS – FUNCTION BLOCKS (FB) FOR PROCESS CONTROL AND ELECTRONIC DEVICE DESCRIPTION LANGUAGE (EDDL) –

Part 4: EDD interpretation

FOREWORD

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

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

This second edition cancels and replaces the first edition published in 2015. This edition constitutes a technical revision.

This edition was developed by merging material from multiple variants of existing EDDL specifications including those from FieldComm Group (Foundation™ Fieldbus[1], HART®[2]), PROFIBUS™[3] Nutzerorganisation e.V. (PNO), and ISA100_Wireless™[4] Compliance Institute (ISA100 WCI). When a profile deviation exists, it is now indicated in the context where the related deviation is found. As a result, the formatting and numbering of this edition may be different from any of the individual specifications from which this edition was derived.

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

- communication profiles ISA100 and GPE were added;
- description of rules for optimized-column-width layout have been added;
- description of the concatenation of labels and help was added;
- color banding for meter type charts was added.

The text of this International Standard is based on the following documents:

CDV	Report on voting
65E/633/CDV	65E/690/RVC

Full information on the voting for the approval of this International 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 *Devices and integration in enterprise systems – 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.

---

1 FOUNDATION™ Fieldbus is the trademark of FieldComm Group. 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 FieldComm Group. 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.

4 ISA100_Wireless™ is the trademark of ISA100 Wireless Compliance Institute. 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.

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

- 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.

EDDL features are limited by profile for each of the communication technologies. The descriptions in this part of IEC 61804 refer to these features in a general sense and not all communication technologies will support all of the features described. The profile definitions in IEC 61804-3 are referred to in order to understand the features supported by each communication technology.

DEVICES AND INTEGRATION IN ENTERPRISE SYSTEMS – 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 are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IEC 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/IEEE 8802-3*

IEC 61804-3, *Devices and integration in enterprise systems – Function blocks (FB) for process control and electronic device description language (EDDL) – Part 3: EDDL syntax and semantics*

IEC 61804-5, *Devices and integration in enterprise systems – Function blocks (FB) for process control and electronic device description language (EDDL) – Part 5: EDDL Builtin library*

IEC 62734, *Industrial networks – Wireless communication network and communication profiles – ISA 100.11a*

IEC 62769-4[5], *Field Device Integration (FDI) – Part 4: FDI Packages*

IEC 62769-7[6], *Field Device Integration (FDI) – Part 7: FDI Communication devices*

3 Terms, definitions, abbreviated terms acronyms and conventions

3.1 General terms and definitions

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

[5] Under preparation. Stage at the time of publication: IEC RFDIS 62769-4:2020.

[6] Under preparation. Stage at the time of publication: IEC RFDIS 62769-7:2020.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

- IEC Electropedia: available at http://www.electropedia.org/
- ISO Online browsing platform: available at http://www.iso.org/obp

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
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
CS	Communication Server
EDD	Electronic Device Description
EDDL	Electronic Device Description Language
FDI	Field Device Integration
FF	Foundation Fieldbus
GPE	Generic Protocol Extension
HART	Highway Addressable Remote Transducer
ISA100	ISA100_Wireless™
PB	PROFIBUS
PC	Personal computer
PI	PROFIBUS and PROFINET International
PI PROFILE PA	PI specific profile for Process Automation field devices
PN	PROFINET

3.4 Conventions

There exists some differences using and interpreting EDDL based on the used communication network and device model. Annex C includes a description of each of the EDDL profiles.

The different communication networks used within this part of IEC 61804 are:

- HART (according to IEC 61784-1 CPF9)
- FOUNDATION fieldbus (according to IEC 61784-1 CPF1)
- PROFIBUS (according to IEC 61784-1 CPF3)
- PROFINET (according to IEC 61784-2 CPF3)
- Communication Server
  - IEC 62769-4 FDI Technical Specification: Part 4: FDI Packages
  - IEC 62769-7 FDI Technical Specification: Part 7: Communication Devices
- ISA100_Wireless (according to IEC 62734)
- GPE (Generic Protocol Extension)
  - IEC 62769-4 FDI Technical Specification: Part 4: FDI Packages
  - IEC 62769-7 FDI Technical Specification: Part 7: Communication Devices.

EDD examples in this document 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 61804-2:2004. 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 document 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 document shall be followed. Annex B lists predefined identifiers that may be used by EDD applications to access standardized information.

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). In order to fit in a limited display space, a handheld application may render all menus using STYLE TABLE. Handheld applications shall render references to images in MENUs of STYLE TABLE always as hyperlink.

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*	TABLE	Legacy root menu prefix for BLOCK_A MENU_ITEMs for FOUNDATION fieldbus and ISA100_Wireless devices e.g. Menu_Top_TB.
offline_root_menu	TABLE	Offline configuration. Handhelds may optionally support this menu.
hh_process_variables_root_menu	TABLE	Process variable views for FOUNDATION fieldbus and HART devices.
hh_device_root_menu	TABLE	Device feature views for set up for FOUNDATION fieldbus and HART devices.
hh_diagnostic_root_menu	TABLE	Diagnostic views for FOUNDATION fieldbus and HART devices.

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 and ISA100_Wireless. HART EDDs shall have root_menu, in addition to all root menus in Table 2. GPE shall have at least one Online menu. GPE shall have the offline_root_menu. Communication Server shall have at least one Online menu.

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
FF, ISA100	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, in STYLE TABLE.
PB, PN	Menu_Main_Specialist if exists or Menu_Main_Maintenance. These menus include online and offline sub-menus. These would be rendered as STYLE MENU.

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 variables root menu

The process_variables_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
FF, ISA100	BLOCK_A PARAMETERS
HART	MENU upload_variables
PB, PN	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, ISA100_Wireless, PROFIBUS, PROFINET, Communication Server profile and GPE profile 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.

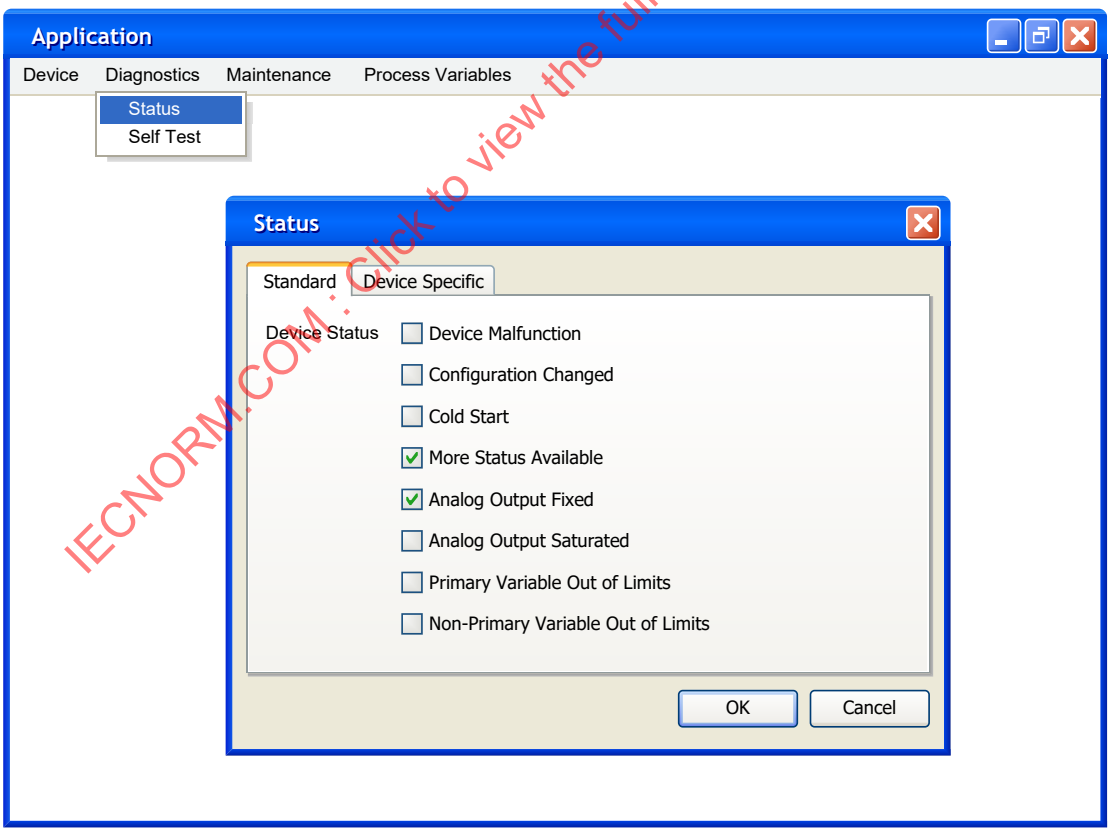


Figure 2 – Example of an EDD application for diagnostics

4.3.5.2 Process variables

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

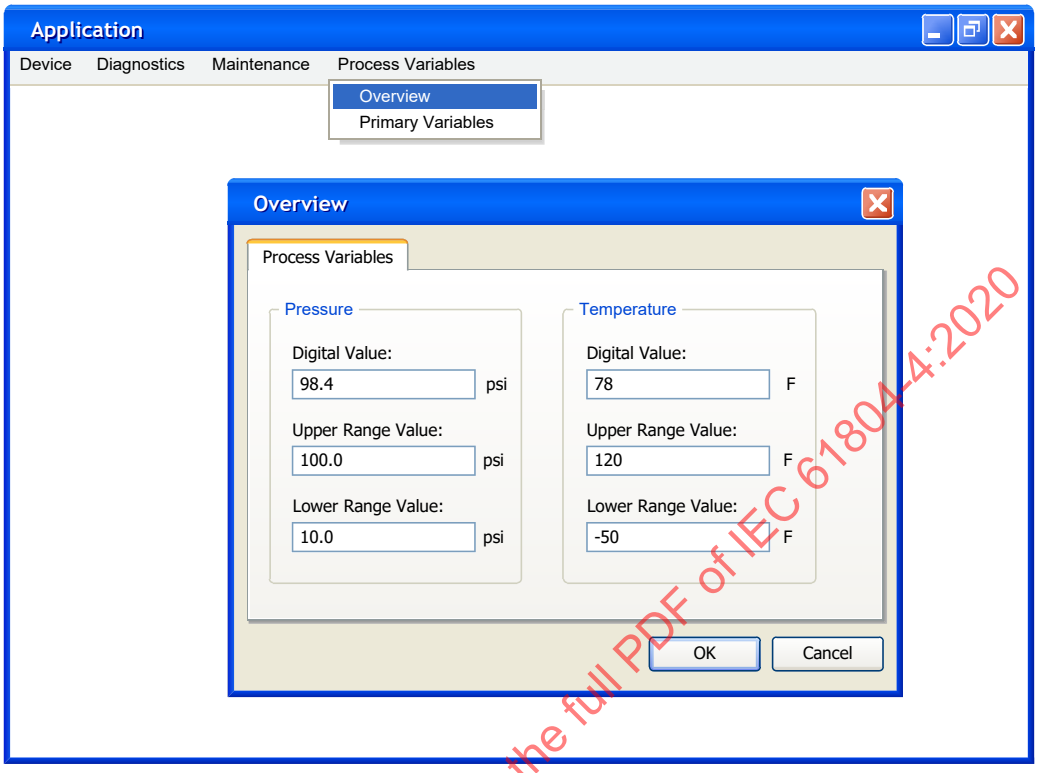


Figure 3 – Example of an EDD application for process variables

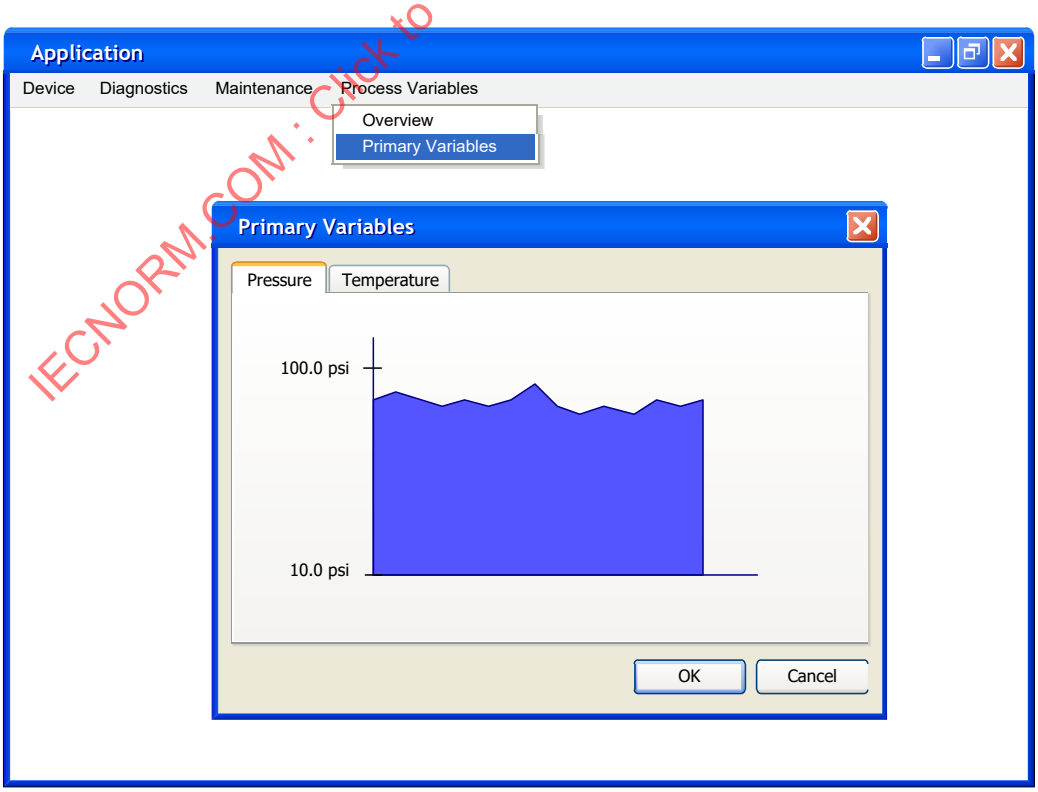


Figure 4 – Example of an EDD application for primary variables

4.3.5.3 Device features

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

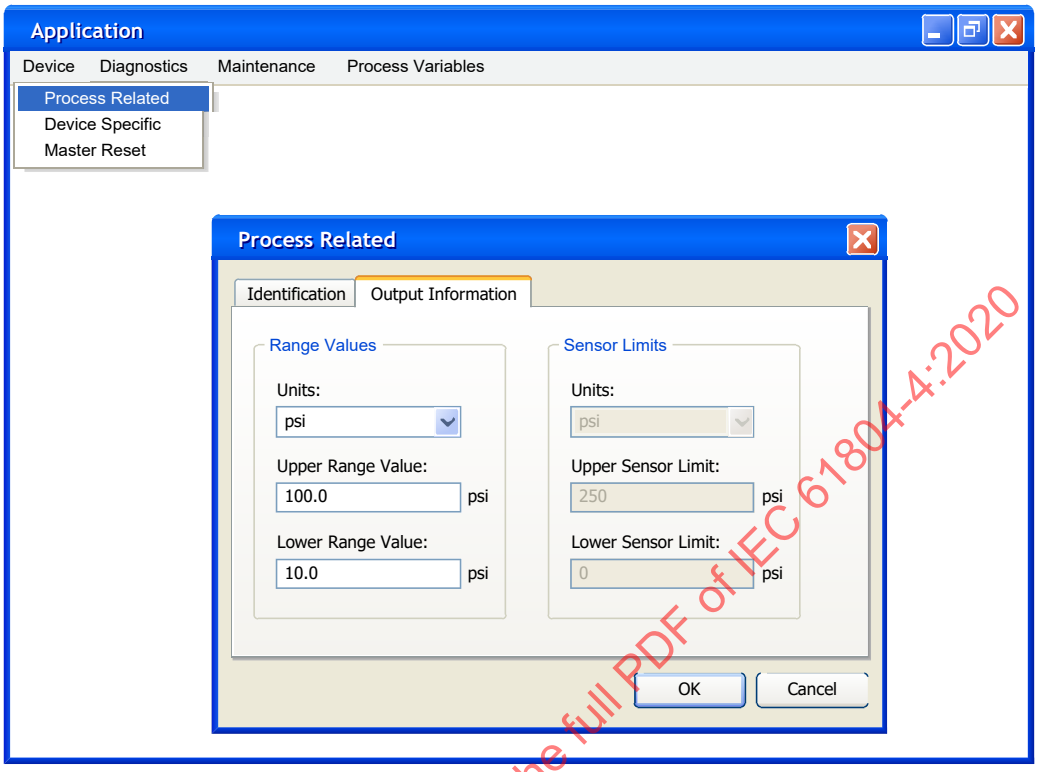


Figure 5 – Example of an EDD application for process-related device features

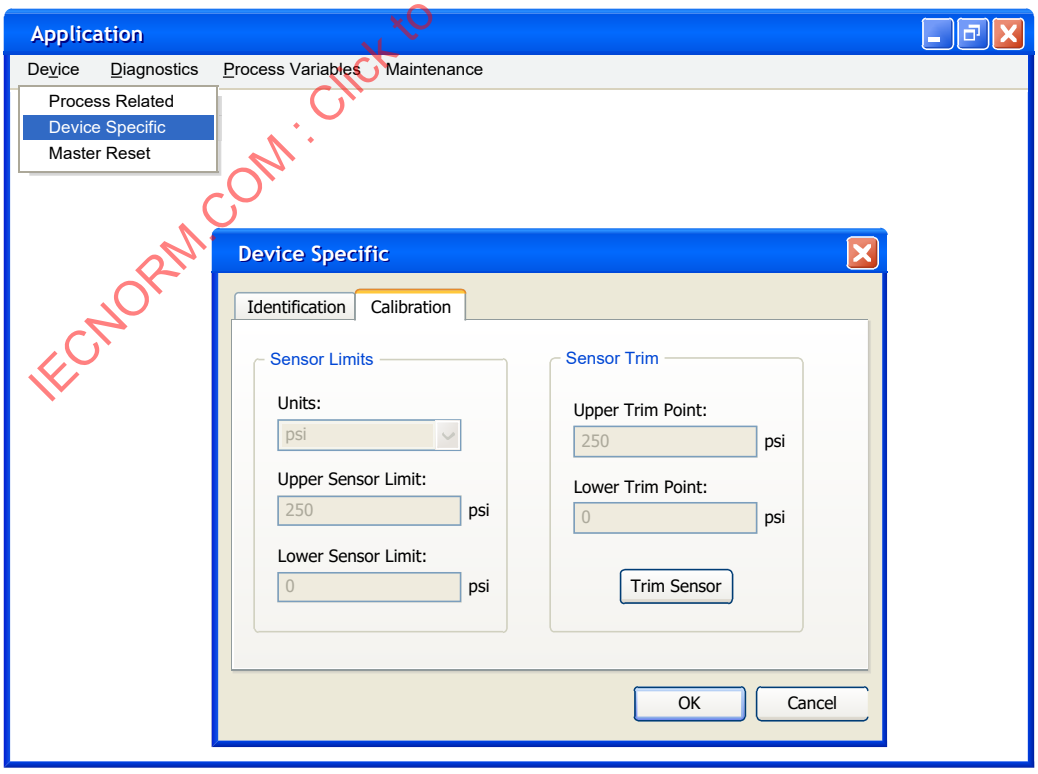


Figure 6 – Example of an EDD application for device features

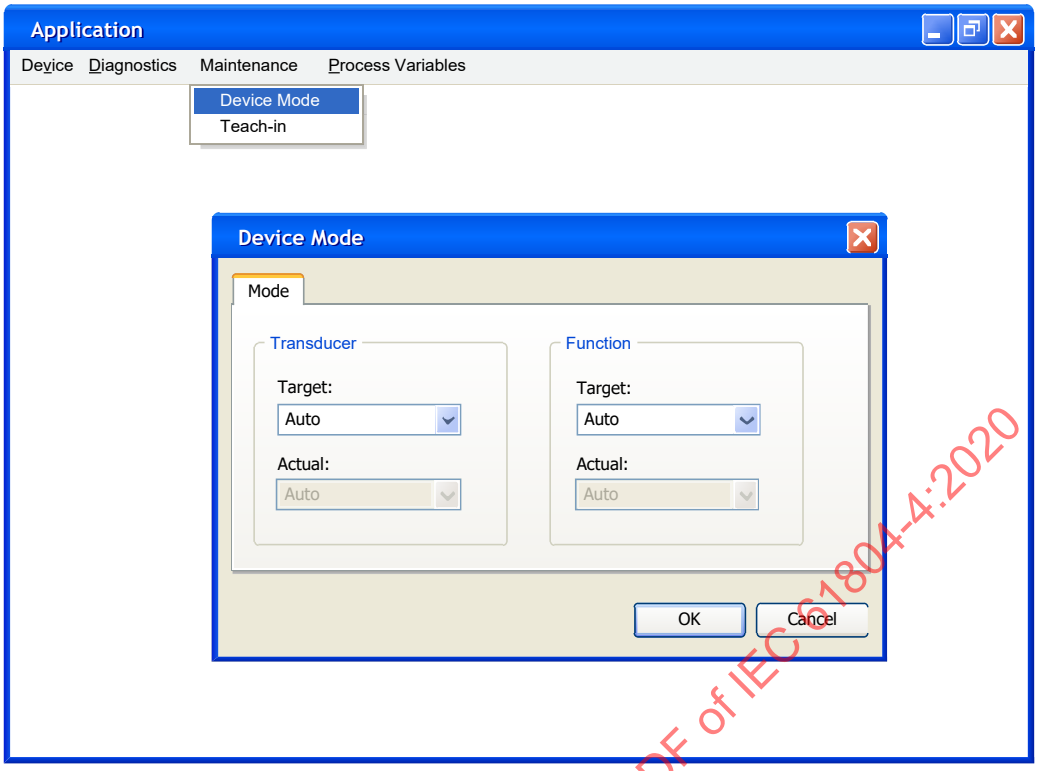


Figure 7 – Example of an EDD application for maintenance features

4.4 Label concatenation for indirect variable references

4.4.1 General

For indirect variable-references only, the LABEL displayed shall be a concatenation of the strings found in the EDD item containers through which the VARIABLE was referenced. This occurs, for example, when a VARIABLE is referenced via a COLLECTION MEMBER or ARRAY ELEMENT. For example, PV.DEVICE_VARIABLE.DIGITAL_VALUE is an indirect reference to a VARIABLE via both arrays and collections. A reference is defined as:

```
reference:= [ identifier | reference [ expression ] |
              reference . identifier ]
```

To specify Label concatenation, variable-references will be classified as a direct, simple, or complex reference.

A "direct reference" is:

```
direct-reference:= identifier
```

String concatenation is not applicable to direct references.

A "simple reference" is:

```
simple-reference:= [ identifier [ expression ] | identifier . identifier ]
```

And a "complex reference" is:

```
complex-reference:= [ simple-reference . identifier |
                      simple-reference [ expression ] |
                      complex-reference . identifier |
                      complex-reference [ expression ] ]
```

Label concatenation is only applicable to simple and complex (indirect) variable-references. Direct variable-references do not have concatenated Labels.

In the following subclauses, any label or description that holds an empty string (“”) shall be treated as if it does not exist.

4.4.2 Simple variable references

For simple variable-references, the Label and Description of the simple variable-reference are space concatenated together. See the summary in Table 5 and Table 6.

Table 5 – Label rule summary for simple variable references

Reference type	Label + description exists	Only label exists	Only description exists	Neither label nor description exists
FILE REFERENCE_ARRAY COLLECTION	Label shall be the space concatenation of the term's Label and the term's Description.	Label shall be the term's Label.	Label shall be the term's Description.	Same Label as direct variable-reference.
RECORD	Label shall be the term's Description.	Label shall be the term's Label.	Label shall be the term's Description.	Same Label as direct variable-reference.

Table 6 – Label rule summary for simple variable references

Reference type	Label exists	
	Yes	No
LIST VALUE_ARRAY	Label shall be Label "[n]" where n is the value of the index.	Label shall be "[n]" where n is the value of the index.

4.4.3 Complex variable references

The LABEL for a complex variable-reference is constructed by space concatenating the Prefix, Body and Suffix strings together.

Prefix-string

The LABEL Prefix-string is generated based on the first term in the complex variable-reference. In the case of a file, reference-array, block, record, or collection reference, the first non-null Description-string is used. See the Prefix rules summarized in Table 7 and Table 8.

Notice that if a FILE, REFERENCE_ARRAY, BLOCK or COLLECTION reference is the first term in the complex variable-reference and it does not have a corresponding Description-string, the search for a Prefix-string continues with the next term in the complex variable-reference. This process is repeated until the Prefix-string is generated or final term in the complex variable-reference is encountered. If the Prefix-string is generated the process continues with the construction of the Body and Suffix strings.

If the final term is encountered with no Prefix-string being generated then the rules for simple variable-references are used to generate the LABEL using the final term in the complex variable-reference.

Table 7 – Prefix rule summary for complex variable references

Reference type	Description exists	
	Yes	No
FILE REFERENCE_ARRAY COLLECTION RECORD BLOCK_A	Prefix is Description.	Skip this term. Continue to next term in the complex variable-reference to locate Prefix string.

Table 8 – Prefix rule summary for complex variable references

Reference type	Label exists	
	Yes	No
LIST VALUE_ARRAY	Prefix shall be Label "[n]" where n is the value of the index.	Prefix shall be "[n]" where n is the value of the index.

Body-string

The Body-string is a sequential concatenation of strings generated from the intermediate terms in the complex variable-reference. See Table 9 and Table 10 for a summary of the rules for constructing the Body-string.

Generation of the Body-string ceases when the final term in the complex variable-reference is reached. The Body-string shall be the null (i.e., "") string if the final term is reached before a Body-string is created. The final term in the complex variable-reference shall be used to generate the Suffix.

Table 9 – Body rule summary for complex variable references

Reference type	
FILE COLLECTION RECORD	Nothing is concatenated to the Body-string.
LIST VALUE_ARRAY	[n] is concatenated to the Body-string where n is the value of the index found in the reference.

Table 10 – Body rule summary for complex variable references

Reference type	Prefix is from a FILE, COLLECTION or REFERENCE_ARRAY	
	Yes	No
REFERENCE_ARRAY	Nothing is concatenated to the Body-string.	[n] is concatenated to the Body-string where n is the index.

Suffix-string

The Suffix-string is generated from the final term in the complex variable-reference. See Table 11 and Table 12 for a summary of the rules for constructing the Suffix-string.

Table 11 – Suffix rule summary for complex variable references

Reference type	
LIST VALUE_ARRAY	[n] is Suffix where n is the value of the index.

Table 12 – Suffix rule summary for complex variable references

Reference type	Description exists	
	Yes	No
FILE REFERENCE_ARRAY COLLECTION RECORD	Suffix is Description.	No Suffix (i.e., an empty string).

4.5 Help concatenation

4.5.1 General

The Help-string for a simple or complex variable-reference is generated in a manner similar to that used for the Label-string and depends on whether it is a simple or complex variable-reference.

In the following clauses, any help that holds an empty string (“”) shall be treated the same as if the help does not exist.

4.5.2 Simple variable references

For simple variable-references, the help string is generated using the rules in Table 13 and Table 14.

Table 13 – Help rule summary for simple variable references

Reference type	Container's help + member/element help exists	Only container's help exists	Only member/element help exists	Neither help exists
FILE REFERENCE_ARRAY COLLECTION RECORD	Help shall be space concatenation of both Help-strings.	Help shall be Container's Help.	Help shall be Member/Element Help.	Help shall be same as with direct variable-reference.

Table 14 – Help rule summary for simple variable references

Reference type	Container's help exists	
	Yes	No
LIST VALUE_ARRAY	Help shall be Container's Help.	Help shall be an empty string.

4.5.3 Complex variable references

For a complex variable-reference, in general, there is possibly a Help-string Prefix and Suffix. Unlike Label concatenation, there is no Help-string Body. Basically, the Help string is the first-found Help space-concatenated to the Help from the final term in the complex variable-reference. If both the Help Prefix and Suffix are empty strings then the Help-string shall be an empty string.

The Help Prefix is generated based on the first term in the complex variable-reference. In the case of a FILE, REFERENCE_ARRAY, or COLLECTION reference, the first non-empty Member/Element Help is used. See the Prefix rules summarized in Table 15 and Table 16.

If the final term is encountered with no Prefix being generated then the rules for simple variable-references are used to generate the Help based on the final term in the complex variable-reference.

Table 15 – Help prefix rule summary for complex variable references

Reference type	
LIST VALUE_ARRAY	Prefix is Help string from the list or value-array (possibly empty).

Table 16 – Help prefix rule summary for complex variable references

Reference type	Member/element help exists	
	Yes	No
FILE REFERENCE_ARRAY COLLECTION RECORD	Help Prefix is Member/Element Help string.	Skip this reference. Continue to next reference to locate HELP Prefix string.

The Help suffix is generated from the final term in the complex variable-reference (see Table 17 and Table 18).

Table 17 – Help suffix rule summary for complex variable references

Reference type	
LIST VALUE_ARRAY	Suffix is Help string from referenced VARIABLE (possibly empty).

Table 18 – Help suffix rule summary for complex variable references

Reference type	Member/element help exists	
	Yes	No
FILE REFERENCE_ARRAY COLLECTION RECORD	Help Suffix is Member/Element Help string.	No Suffix (i.e., an empty string).

4.6 Containers and contained items

4.6.1 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.

4.6.2 Permitted and default STYLEs

Table 19 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.

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 19 – Permitted contained items and default STYLES

Container (Parent)	Permitted contained items (Child items)															Default STYLE of contained items (used if STYLE is not defined or in case of conflicts)
	MENU	DIALOG, WINDOW	PAGE	GROUP	TABLE	EDIT_DISPLAY	VARIABLE	COLLECTION, RECORD	ARRAY, LIST	CHART, GRAPH, GRID	IMAGE	static text	METHOD	PLUGIN	COLUMNBREAK, ROWBREAK	SEPARATOR	
MENU	✓	1	X	X	10	✓	8	D	8	2	2	✓	✓	✓	6	6	**Communication profiles: CS, FF, GPE, ISA100, PB, PN** MENU if no data item is in the menu, otherwise DIALOG. **Communication profiles: HART** MENU
DIALOG, WINDOW	✓	1	✓	✓	7	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	5	PAGE
PAGE	✓	1	X	✓	7	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	5	GROUP
GROUP	✓	1	X	✓	7	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	5	GROUP if the parent of the parent GROUP is not of STYLE GROUP. DIALOG rendered as button or hyperlink if the parent of the parent GROUP is of STYLE GROUP.
TABLE	X	9	X	X	✓	✓	3	D	✓	2	3	✓	✓	✓	6	6	TABLE

Key

✓ Shall be rendered as defined for the contained item; e.g., for a “MENU” according to its STYLE attribute.

X Not allowed. Default STYLE shall be used to solve this conflict (see column “Default STYLE of contained items”).

D Default STYLE shall be used (see column “Default STYLE of contained items”).

1 Shall be rendered as button or hyperlink by using the LABEL. The item shall be presented when the button or hyperlink is selected.

2 Shall be rendered as a button or hyperlink by using the LABEL. The item shall be presented on a dialog (i.e., a modal window), when the button or hyperlink is selected.

3 Shall be rendered as button or hyperlink by using the LABEL, or inline. If not rendered inline, then the item shall be presented on a dialog (i.e., a modal window), when the button or hyperlink is selected.

4 Shall be rendered as a button or hyperlink by using the LABEL. The item shall be presented on a modeless window, when the button or hyperlink is selected.

5 Shall be interpreted as COLUMNBREAK.

6 Shall be rendered as a line or ignored.

7 **Communication profiles: CS, FF, GPE, ISA100, PB, PN**

Shall be rendered according to note “✓”.

Communication profiles: HART

Shall be rendered according to note “**X**”.

8 **Communication profiles: CS, FF, GPE, ISA100, PB, PN**

Shall be rendered according to note “**2**”.

Communication profiles: HART

Shall be ignored and not presented.

9 **Communication profiles: CS, FF, GPE, ISA100, PB, PN**

Shall be rendered according to note “**1**”.

Communication profiles: HART

Shall be rendered according to note “**X**”.

10 **Communication profiles: CS, FF, GPE, ISA100, PB, PN**

Shall be rendered according to note “**4**”.

Communication profiles: HART

Shall be rendered according to note “**D**” if subordinate to DIALOG or WINDOW, otherwise according to note “**4**”.

4.6.3 Containers

4.6.3.1 Menu

Communication profiles: CS, FF, GPE, ISA100, PB, PN

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

Communication profiles: HART

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

4.6.3.2 Window

A menu of STYLE WINDOW takes the form of a modeless window. The LABEL of the menu shall be used as the caption for the window.

4.6.3.3 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. The LABEL of the menu shall be used as the caption for the window.

4.6.3.4 Table

Communication profiles: CS, FF, GPE, ISA100, PB, PN

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.

Communication profiles: HART

A menu of STYLE TABLE shall be rendered as a cascaded menu tree. Each item shall be represented as one or multiple rows, e.g. ARRAYs. The submenus build a hierarchy, which shall be shown. The label of the MENU-item (including BIT_ENUMERATED VARIABLEs) shall be displayed. Selecting MENU item shall cause the contents to be displayed.

4.6.3.5 Page

A menu of STYLE PAGE within a WINDOW or DIALOG shall be rendered as a tab and spans the entire canvas width. When multiple pages (tabs) are rendered in the same window or dialog, each tab shall be individually selectable to enable the contents of a particular tab. The LABEL of the menu shall be used as the caption for the tab. There shall be an implicit ROWBREAK before and after the PAGE.

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

COLUMNBREAKs and SEPARATORs between pages shall be ignored.

4.6.3.6 Group

A menu of STYLE GROUP takes the form of a group box. The LABEL of the menu shall be used as the caption for the group. 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 may 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

4.6.4 Contained items

4.6.4.1 Overview

How a container renders contained items is described in 4.6.3.1. 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. However, there shall be some methodology, such as a "tool tip", "pop- up window", “expanding window”, etc., to display the entire string to the user.

4.6.4.2 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.

4.6.4.3 Variables

4.6.4.3.1 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.
- HANDLING = WRITE: the value should not be read from the device due to being referenced as a contained item, e.g. in a MENU.
- HANDLING = READ & WRITE: The value should be editable.
- CLASS = DYNAMIC: the value should be updated continuously with current read values from the device. If the value has been edited, continuous updates shall no longer occur so that the edited value is not overwritten with the value read from the device.

For string type VARIABLEs the value should not be truncated or wrapped in multiple lines. Word wrapping could be confused with a true "new line"characters ('\n') in the string value.

When no VARIABLE value is available to be displayed, e.g. due to a change in VALIDITY or HANDLING, the EDD application shall indicate that the variable is in uninitialized state.

Table 20 shows how the EDD application should indicate the uninitialized state of VARIABLEs.

Table 20 – Uninitialized state of VARIABLEs on user interface

VARIABLE TYPE	Uninitialized state indication
Numeric values ENUMERATED String values Date, time and duration values	Blank with indication of “uncertain” state close to the value field
BIT_ENUMERATED	Indication of no bit set along with indication of “uncertain” state close to the value field
BOOLEAN	Indication of FALSE along with indication of “uncertain” state close to the value field

4.6.4.3.2 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.

When the whole BIT_ENUMERATED VARIABLE is displayed, the display of the bits in the VARIABLE shall appear grouped together.

When a bit enumeration is of CLASS DIAGNOSTIC, then the bits displayed should be treated as a status indicator. A red indication should be presented when the bit is set and green when the bit is reset.

Communication profiles: HART

> When the whole BIT_ENUMERATED VARIABLE is displayed, the display of the bits in the VARIABLE shall be displayed inside a group-box. The LABEL of the BIT_ENUMERATED VARIABLE shall be used as the label of the group box. EDD applications with limited display constraints that prevent the display of group boxes are not required to meet this requirement.

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.

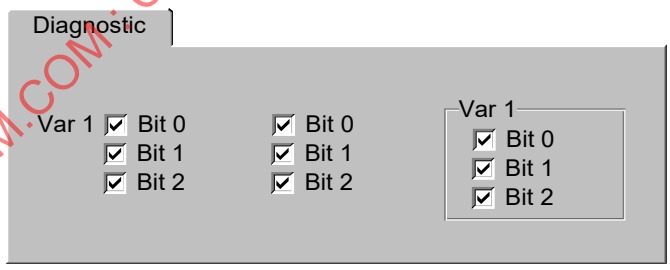


Figure 11 – Example of an EDD application for a variable of type BIT_ENUMERATED

4.6.4.3.3 Variable of TYPE INDEX

When a VARIABLE of type INDEX is presented to the user, the EDD application shall determine the value 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 referenced ELEMENTS exist then this shall be displayed else;
3) the LABEL of the ARRAY or LIST 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.

4.6.4.3.4 Variable with HANDLING WRITE

According to the HANDLING attribute a VARIABLE with HANDLING WRITE is a "write-only" variable. This means that the variable is editable and may be written to the device, but it should not be read from the device.

For online session, EDD application shall display the VARIABLE value as uninitialized (see Table 20), until the VARIABLE is modified during the same edit session.

For offline session, EDD application shall display the VARIABLE value from the current cache.

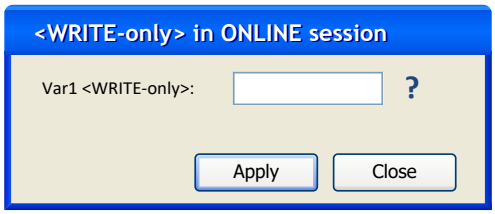
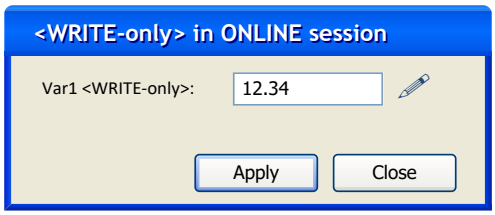
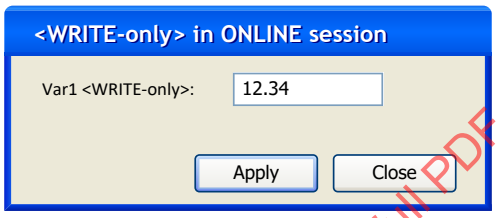
Figure 12 shows an EDD example with a "write-only" variable.

```
//--- UI description

MENU    device_root_menu
{
    LABEL       "Device setup";
    ITEMS
    {
        menu_WriteOnly_online
    }
}

MENU    menu_WriteOnly_online
{
    LABEL       "ONLINE session";
    STYLE       DIALOG;
    ITEMS
    {
        Var1
    }
}


//--- Data description

VARIABLE Var1
{
    LABEL       "Var1 <WRITE-only>";
    CLASS       DEVICE;
    TYPE        FLOAT
    {
        DEFAULT_VALUE 11.22;
    }
    HANDLING    WRITE;
}
```

Figure 12 – EDD example with a "write-only" variable (HANDLING WRITE)

Table 21 shows the operating steps for a "write-only" variable in an online session.

Table 21 – Example of "write-only" variable in an online dialog

Step	User interface	Description
Step 1: The dialog is opened (initial state).		Since "Var1" is a "write-only" variable, the value cannot be read from device. "Var1" is displayed editable and blank with indication of "uncertain" state close to the value field .
Step 2: Edit "Var" = 12.34		The new value of "Var1" is displayed with indication of "changed" state close to the value field.
Step 3: Press "Apply" to commit the edited value		The new value of "Var1" is committed, i.e. transferred to the device. Then the committed value is displayed without indication of "uncertain" or "changed" state.

4.6.4.4 Collections and records

COLLECTIONs and RECORDs shall be displayed as MENU with STYLE GROUP. COLLECTIONs can contain COLLECTIONs. This is handled like nested groups (see 4.6.3.6). The rules about nesting groups apply also in this case.

4.6.4.5 Arrays and lists

Communication profiles: CS, FF, GPE, ISA100, PB, PN

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

Communication profiles: FF

Each element of the array should be displayed as if each array element was specifically specified (in order) as menu items.

Communication profiles: HART

HART does not support arrays and lists references as contained items on a MENU.

Non-existent LIST elements referenced on a MENU should be treated as if their VALIDITY attribute value is FALSE.

4.6.4.6 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 DISPLAY_ITEMS shall be displayed as read only items and EDIT_ITEMS shall be handled like ITEMS from a MENU of STYLE DIALOG.

4.6.4.7 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 4.7 and IEC 61804-3 for information on the effect of the WIDTH attribute on the layout of the GRAPH. 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).

4.6.4.8 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 4.7 and IEC 61804-3 for information on the effect of the WIDTH attribute on the layout of the CHART. For CHARTs that contain a time axis, there shall be an indication of what the time duration is (e.g. seconds, minutes, absolute time, etc.).

4.6.4.9 Images

Handheld applications may display an image referenced on a MENU as a hyperlink. Otherwise, EDD applications shall render images using the following rules:

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

b) When INLINE is not specified, the EDD application shall display an IMAGE in original size. The EDD application shall also insert a ROWBREAK above the IMAGE, if items exist before the IMAGE and insert a ROWBREAK below the IMAGE, if items exist after the IMAGE.

c) When INLINE is specified, the EDD application shall render the IMAGE depending on the LAYOUT_TYPE (see 4.7.2). For "equal-column-width layout" (see 4.7.2.2), the EDD application shall scale down an IMAGE if the IMAGE width is larger than the column width in which the IMAGE is contained. The EDD application shall keep original aspect ratio of IMAGES. For "optimized-column-width layout" (see 4.7.2.3), the EDD application shall not scale down an IMAGE, regardless whether INLINE is specified or not (see 4.7.5.10).

d) The EDD application shall not scale up an IMAGE.

e) If the IMAGE has the LINK attribute defined, this shall be indicated to the user.

f) The EDD application shall display IMAGES having transparency against the background of its container. In this case, the color of the container applies as the background color of the image.

Communication profiles: HART

> The size of an image shall be less than one hundred fifty thousand bytes (150 000) and the size of the sum of all images in a EDD shall not exceed five million bytes (5 000 000).

Images with a width of 210 pixels or less should be able to fit INLINE without any scaling applied. EDD applications with limited display constraints may not satisfy this recommendation.

4.6.4.10 Static text

The text that is referenced by the MENU should be displayed as a non-editable text. In a table view, long text may be truncated or wrapped in multiple lines on the menu. For all other views, the text will be wrapped in multiple lines, if the text is longer than the width of dialog, window, page, group, or column.

4.6.4.11 Grid

A grid should be displayed in a manner that is consistent with the definition of the corresponding GRID. The LABEL of the grid is shown at first and below the grid area. Refer to the layout rules defined in 4.7 and IEC 61804-3 for information on the effect of the WIDTH attribute on the layout of the GRID. Mechanisms should be provided to scroll the grid area, if the width or height is too small to show the complete data.

4.6.4.12 Plugin

A Plugin should be rendered as a button or a hyperlink. The LABEL of the corresponding PLUGIN should appear on the button or as the hyperlink text. When the button or hyperlink is pressed, the corresponding PLUGIN should be invoked.

4.7 Layout rules

4.7.1 Overview

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

The basic elements of a layout are shown in Figure 13 and described in Table 22.

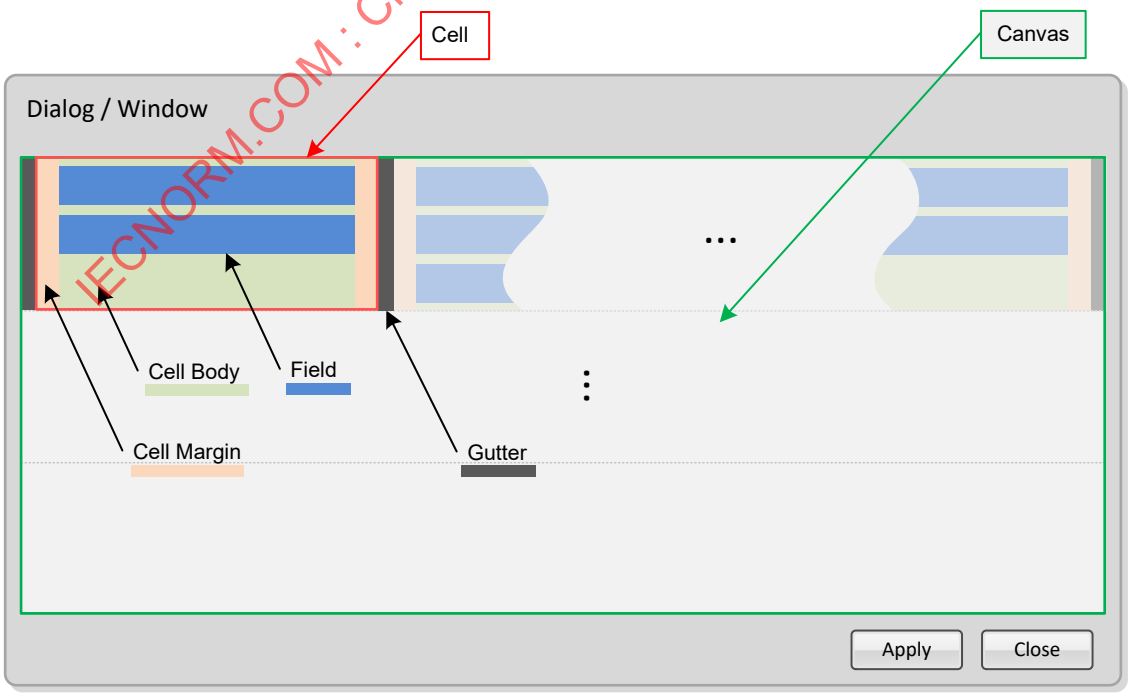


Figure 13 – Basic layout elements

Table 22 – Description of layout content

Layout element	Description and rules
Canvas	Area of WINDOWs, DIALOGs or PAGEs in which the EDD application shall arrange the content of the related MENU. A canvas is organized in rows and columns. The default display area for the canvas shall be wide enough to display at least three columns without horizontal scroll bar. The default display area of the canvas should be tall enough to display at least 15 simple fields in a single column without vertical scrolling. EDD applications with limited display constraints may not satisfy this recommendation. When two or more rows have the same number of columns, the column breaks in those rows shall be aligned. The organization of columns in the canvas shall be determined by the attributes of the construct LAYOUT_TYPE, COLUMNWIDTH_EQUAL or COLUMNWIDTH_OPTIMIZED
Cell	Area between row breaks and column breaks containing fields.
Field	Area that a menu item occupies.
Simple field	Area that would be occupied in a cell by a numeric VARIABLE, its label, its unit, and status indication (if displayed). All simple fields in a cell shall be displayed with All visible labels aligned Values aligned All visible units aligned All visible status aligned
Cell body	Area occupied by the collection of all fields in a cell. The organization of cell bodies in a row shall be determined by the attributes of the construct LAYOUT_TYPE, COLUMNWIDTH_EQUAL or COLUMNWIDTH_OPTIMIZED
Cell margin	The white space between the cell boundary and the cell body All cell margins in a row shall be equal
Gutter	The white space between any two cells and between a cell and the edge of the canvas in a given row All gutters in a row shall be equal

4.7.2 Controlling the layout by LAYOUT_TYPE attribute

4.7.2.1 Overview

The LAYOUT_TYPE attribute is part of the EDD identification information and specifies how the EDD application shall render all graphical user interfaces defined in the EDD.

4.7.2.2 Equal-column-width layout (COLUMNWIDTH_EQUAL)

When the LAYOUT_TYPE is defined as COLUMNWIDTH_EQUAL, each MENU with STYLE evaluated as WINDOW, DIALOG, PAGE, or GROUP shall be rendered using equal-width columns.

Figure 14 shows an example of a layout with equal column widths in each row.

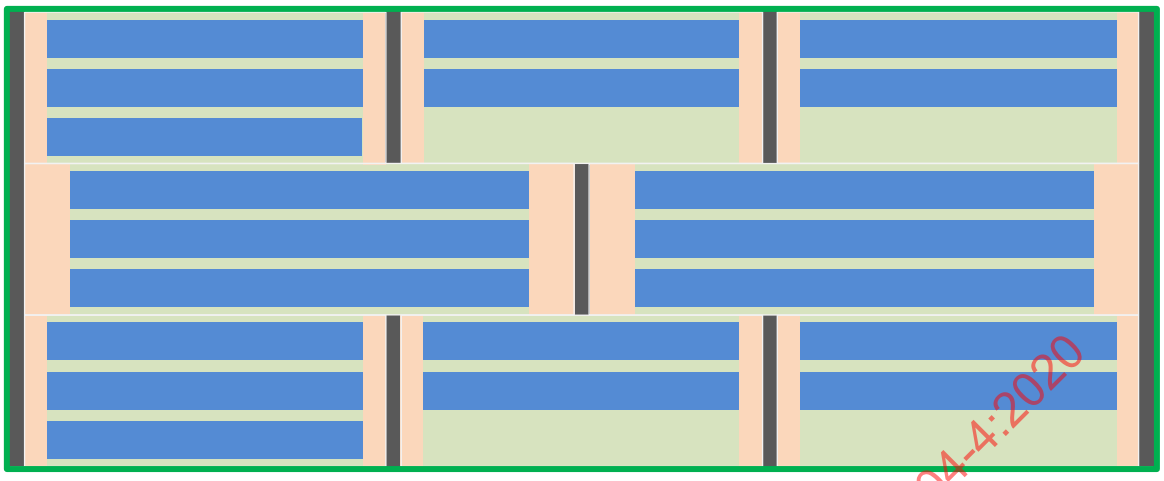


Figure 14 – Example of layout with equal column width

The width of each column in any row shall be the same as each other.

4.7.2.3 Optimized-column-width layout (COLUMNWIDTH_OPTIMIZED)

When the LAYOUT_TYPE is defined as COLUMNWIDTH_OPTIMIZED, each MENU with STYLE evaluated as WINDOW, DIALOG, PAGE, or GROUP shall be rendered using width-optimized columns.

Figure 15 shows an example of a layout with optimized column widths according to the content.

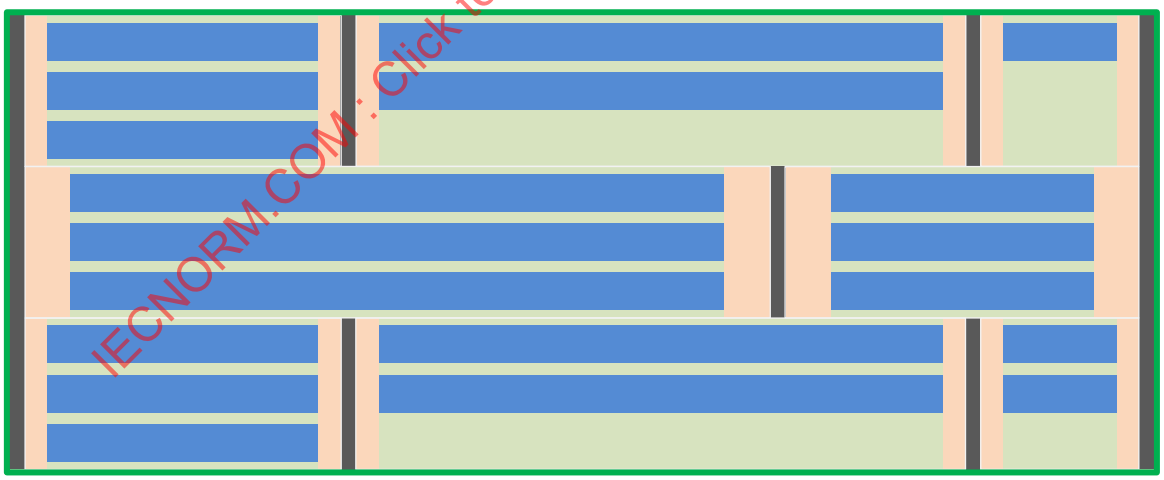


Figure 15 – Example of layout with optimized column width

The EDD application shall arrange the layout depending on the actual number of COLUMNBREAKs and the width of the containers. The width of each column shall be optimized according to the widest menu item in the column. All rows with the same number of columns shall be considered for the calculation.

In contrast to the equal-column-width layout, menu items will not be moved automatically to the next row when a menu item would span to a column higher than the 5th. To start a new layout row in an optimized-column-width layout, a ROWBREAK is required.

Figure 16 shows a layout cell containing a constant string (a), a read-only VARIABLE (b), a METHOD (c) rendered as button, and an editable VARIABLE (d). The labels of the VARIABLEs are displayed to the left. The cell divided in separate areas for the label, the input field containing the value, the variable status and the unit.

Constant strings, menu items rendered as button, and the VARIABLE values shall be aligned and displayed in the input field area.

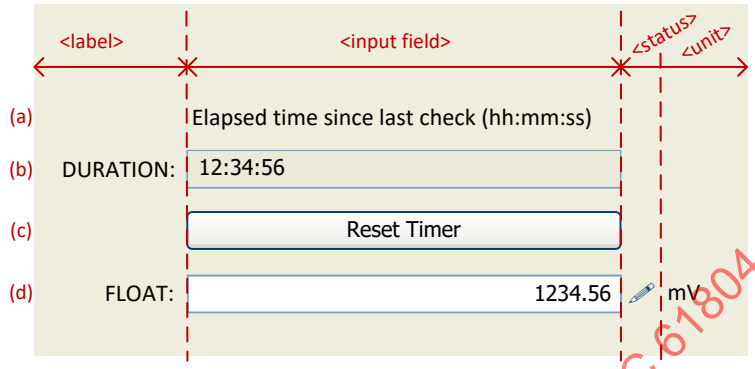


Figure 16 – Cell body in a layout with optimized column width (label to the left)

Figure 17 shows a layout cell containing the same menu items like in Figure 16, but the labels of the VARIABLEs are displayed on top. Label area and input field area are merged.

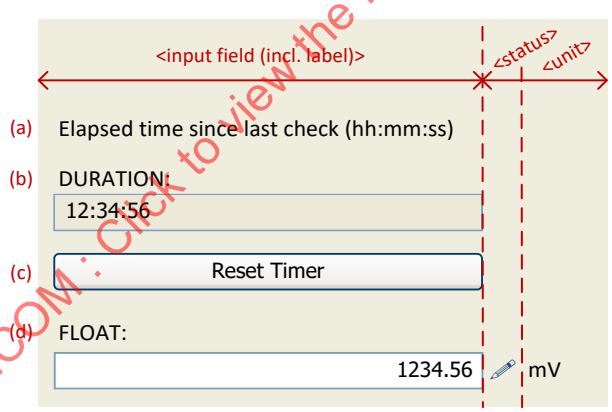


Figure 17 – Cell body in a layout with optimized column width (label on top)

Table 23 summarizes the minimum and maximum width limits for input fields according to the menu item type.

Table 23 – Minimum and maximum width for input fields spanning one column

Layout element	Minimum / maximum width for input fields (approx. pixel values)
Menu items rendered as button	Min. 80 pixels / no max. limit
Constant strings	No min. limit / max. 350 pixels
VARIABLEs	Min. 100 pixels (may be wider depending on the UI control) / max. 350 pixels

The calculation of the optimized column width shall be made in the following steps:

1) The required width of each menu item shall be determined according to the following rules:
   - For menu items rendered as button or hyperlink, e.g. METHODs, the width shall be determined by the label.
   - For constant strings, the width shall be determined by the actual string length. For very long strings exceeding the max. width of an input field, automatic word wrapping shall be supported.
   - For VARIABLEs, the width shall be calculated by totalizing the requested width for the label, for the value and unit and, if displayed, for the variable status. The width of the input field displaying the value depends on the variable's data type. For ENUMERATED or BIT_ENUMERATED variables, the width shall be determined by the widest enumeration element.
     Since in a layout showing the label on top, the label area and input field area are merged, the input field area should be determined considering both, the width of the label and the width of the input field. When the label is wider than the requested width of the input field, the input field should be expanded according to width of the label. Label and unit should not be truncated.
   - For IMAGEs, the width shall be determined by the original width of the image and shall not be adjusted, regardless of whether or not the image referenced is marked with INLINE qualifier. If the IMAGE requires more width than available, the column shall be expanded.
   - For items rendered as GROUP, the width shall be determined according to the content considering the menu label and the margins in the group box. If the menu label is wider than the content, the width of the group box shall be determined according to the menu label.
2) Further, the minimum and maximum width limits for the input field as summarized in Table 23 shall be regarded. Note that in layouts showing the label on top, the input field may be wider than the maximum width limit.
3) The actual column width shall be evaluated as the width of the widest item in the column.
4) To provide a well-arranged layout, the items in each column shall be expanded to span the whole column by expanding the respective input fields (see Figure 16).
   For menu items not rendered with an input field, the full area of the menu item shall be expanded.
5) In a multi-row and -column layout, the number of columns may vary from row to row. The content of rows with a smaller number of columns shall be expanded, so that each row spans the same width. In this case, the input fields may be wider than the respective maximum width specified in Table 23.
6) After calculating the width of each column, the resulting width of the canvas shall be calculated by totalizing the width of each column. When the resulting width of the canvas is wider than the width of the screen, a horizontal scrollbar shall be provided.

When the user changes the size of the canvas by resizing the dialog or window, the following rules shall apply:

- When shrinking, the calculated optimized column width shall apply as the minimum width of each column.
- When expanding, the width gain shall be distributed evenly among all columns.
- The content of each column shall always span the whole column by adjusting the width of the respective input field.

When the required width changes due to a conditional expression, the resulting layout may grow, not shrink. If the resulting layout would require less width, the excess width shall be distributed evenly among all columns.

4.7.3 Layout rules for WIDTH and HEIGHT

4.7.3.1 General

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 simple field, see Table 24. The default value of WIDTH and HEIGHT is MEDIUM for CHART, GRAPH, GRID; XXX_SMALL for VARIABLE height, and width.

Equal-column-width layout:

> 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 behavior as an explicitly defined ROWBREAK.

Optimized-column-width layout:

> The canvas should be divided depending on the number of COLUMNBREAKs and the width of the containers. Columns may span to a column higher than the 5th without being moved to the next row through an implied ROWBREAK.

The WIDTH and HEIGHT attribute define how many columns the item spans and the height in multiples of minimum height possible for a simple field e.g. variable, constant string, menu and method reference, see Table 24.

Table 24 – 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

WIDTH and HEIGHT value	WIDTH	HEIGHT	Applicability
LARGE	4	9	CHART GRAPH GRID VARIABLE
X_LARGE	5	11	CHART GRAPH GRID VARIABLE
XX_LARGE	5	13	CHART GRAPH GRID VARIABLE

4.7.3.2 VARIABLEs spanning columns by WIDTH (optimized-column-width layout)

The WIDTH attribute of an item specifies the number of columns the item spans. Thereby the following rules shall apply:

1) When spanning normal, width consuming columns, the menu item shall be expanded to span all corresponding columns (see Figure 44).
2) When spanning empty or very narrow columns, the width increment of this menu item shall be divided evenly on all corresponding columns. The EDD application shall apply width increment steps considering the minimum width of input fields (see Table 23).
3) The empty or narrow columns shall be expanded accordingly (see Figure 19).
4) For VARIABLEs spanning more than one column, only the input field shall be expanded.

Figure 19 shows a layout containing 2 rows, the 1st with 5 narrow columns and the 2nd with variables spanning a different number of columns by defining the WIDTH attribute. Figure 18 shows the corresponding source code.

```
MENU menu_Layout_Width_on_Variables
{
        LABEL   "WIDTH on Variables";
        STYLE   DIALOG;
        ITEMS
        {
                "1",    COLUMNBREAK,            // 5 "narrow" columns
                "2",    COLUMNBREAK,
                "3",    COLUMNBREAK,
                "4",    COLUMNBREAK,
                "5",
                ROWBREAK,
                var_ASCII,                      // ASCII variable with no WIDTH → default
                var_ASCII_Width_X_SMALL,        // ASCII variable with 'WIDTH X_SMALL'
                var_ASCII_Width_SMALL,          // ASCII variable with 'WIDTH SMALL'
                var_ASCII_Width_MEDIUM,         // ASCII variable with 'WIDTH MEDIUM'
                var_ASCII_Width_LARGE,          // ASCII variable with 'WIDTH LARGE'
                var_ASCII_Width_X_LARGE         // ASCII variable with 'WIDTH X_LARGE'
        }
}

//--- Exemplary VARIABLE definition
VARIABLE var_ASCII_Width_X_SMALL
{
        LABEL   "ASCII (X_SMALL = 1C)";
        CLASS   LOCAL;
        TYPE    ASCII(20);
}
```

Figure 18 – EDD source code for a layout with VARIABLEs spanning columns

The width of the 1st column is determined by totalizing the width of

- the widest label, which is "ASCII (MEDIUM = 3C)",
- the value of the 1st variable "ASCII (Default = 1C)",
- no unit to be considered.

The width of column 2 to 5 is determined by the width increment steps due to spanning empty or narrow columns.

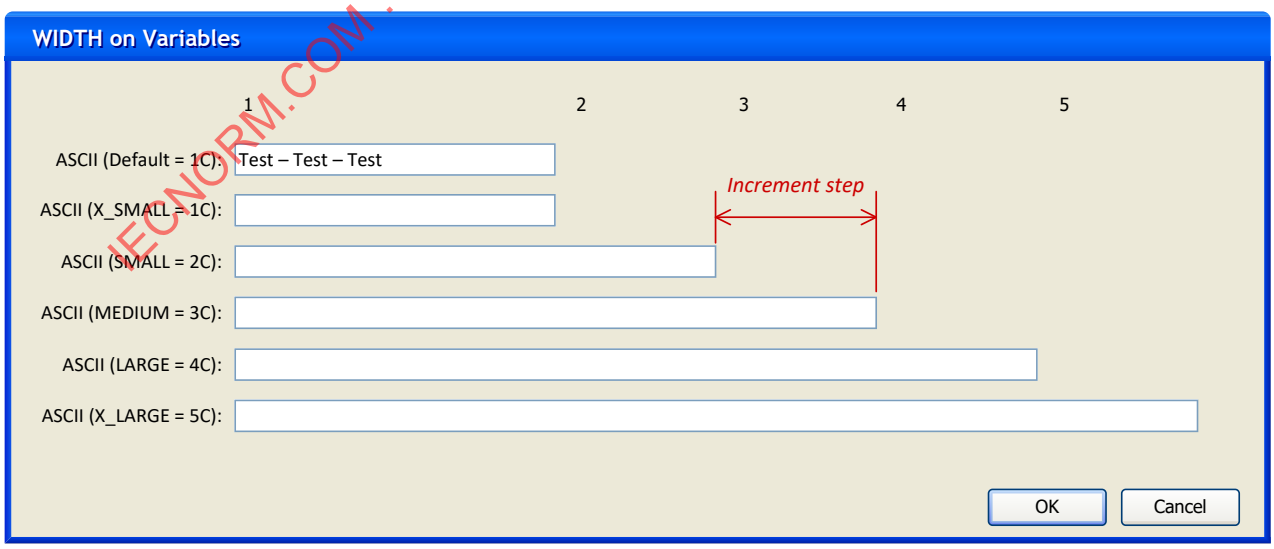


Figure 19 – Layout with VARIABLEs spanning multiple columns

4.7.3.3 CHART and GRAPH spanning columns by WIDTH (optimized-column-width layout)

Depending on the CHART or GRAPH declaration in the EDD, the EDD application should define a minimum width regarding the width for structuring elements (e.g. y-axes, legends) and the characteristics of the UI control used to represent the CHART or GRAPH.

To minimize the need of a horizontal scrollbar, the EDD application should define a maximum width for a CHART or GRAPH spanning 5 columns considering the actual screen characteristics.

The resulting increment step as a minimum width gain per column is: (Max. width – min. width) / 4.

The WIDTH attribute of an item specifies the number of columns the CHART or GRAPH spans. Thereby the following rules shall apply:

1) When spanning empty or very narrow columns, the width gain of the CHART or GRAPH shall be divided evenly on all corresponding columns. The EDD application shall apply a number of the width increment steps according to the WIDTH attribute.
2) When spanning columns consuming more width than one increment step, the CHART or GRAPH shall be expanded to span all corresponding columns.
3) The maximum width may be exceeded to make all layout rows span the same width.

4.7.3.4 GRID spanning columns by WIDTH (optimized-column-width layout)

To minimize the need of a horizontal scrollbar, the EDD application should define a maximum width for a GRID spanning 5 columns considering the actual screen characteristics and the corresponding increment step as a minimum width gain per column.

The WIDTH attribute of an item specifies the number of columns the GRID spans. Thereby the following rules shall apply:

1) When spanning empty or very narrow columns, the width gain of the GRID shall be divided evenly on all corresponding columns. The EDD application shall apply a number of the width increment steps according to the WIDTH attribute.
2) When spanning columns consuming more width than one increment step, the GRID shall be expanded to span all corresponding columns.
3) The EDD application shall consider the actually required width for the GRID depending on the content (VECTORS) as the maximum width of the GRID. This width may be exceeded to make all layout rows span the same width.

4.7.4 Layout rules for COLUMNBREAK and ROWBREAK

4.7.4.1 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 at the top of the next column in the same row. 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 20 for EDD source code example and Figure 21 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,
                COLUMNBREAK,
                Element12,
                Element13,
                Element14
        }
}
```

Figure 20 – EDD source code for layout for protruding elements example

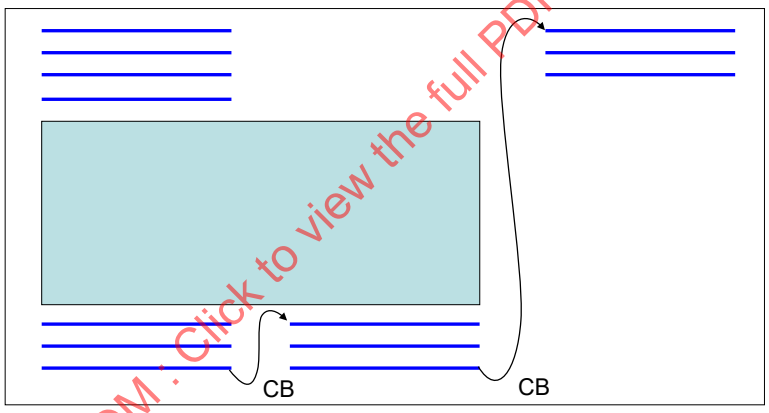


Figure 21 – Layout for protruding elements

4.7.4.2 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 next 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 22 for EDD source code example and Figure 23 for the corresponding layout.

```
MENU menu1
{
        LABEL "menu 1";
        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,
                COLUMNBREAK,
                Element15,
                Element16,
                Element17,
        }
}
```

Figure 22 – EDD source code for layout for partially filled rows example

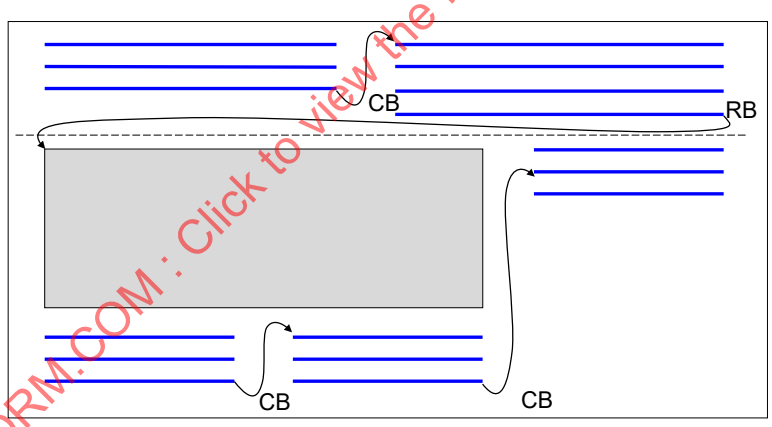


Figure 23 – Layout for partially filled rows

Figure 25 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 24 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 24 – EDD source code for layout for partially filled rows example

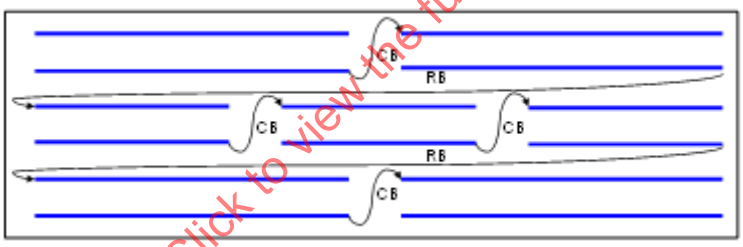


Figure 25 – Layout for partially filled rows

4.7.4.3 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).

With the following example, the layout for oversized elements should be explained for both layout types based on the EDD source code example shown in Figure 26.

```
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 26 – EDD source code for layout for oversized elements example

Equal-column-width layout:

Due to the maximum of 5 columns for equal-column-width layout, the oversized element will be moved to the 1st column of the next row (see Figure 27).

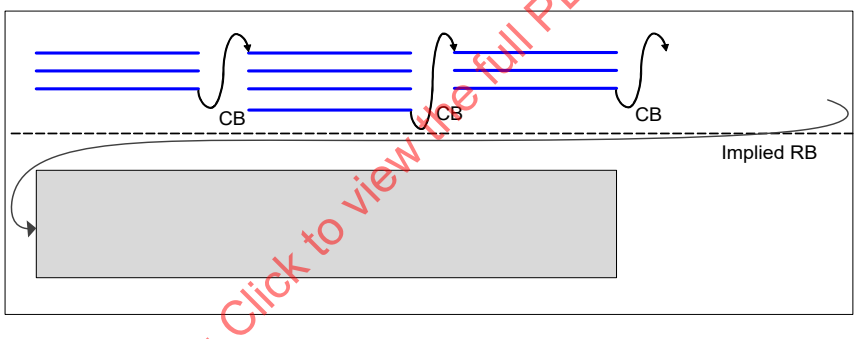


Figure 27 – Oversized element in a layout with equal column width

Optimized-column-width layout:

Since the maximum number of 5 columns does not apply for optimized-column-width layout, the oversized element will not be moved to the next row, but to the next column in the same row to span column 4 up to 6 (see Figure 28).

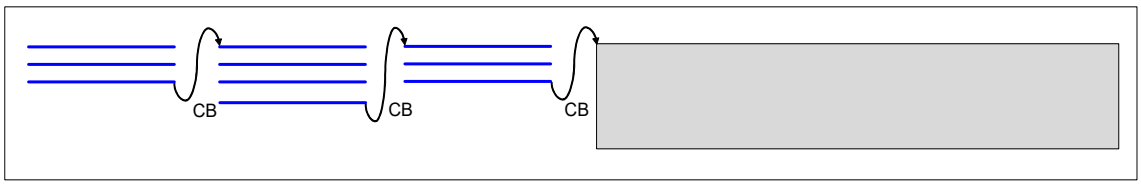


Figure 28 – Oversized element in a layout with optimized column width

To move the oversized element into the 1st column of the next row like shown in Figure 27, the EDD developer shall use a ROWBREAK instead of a COLUMNBREAK to control the position of the oversized element.

4.7.4.4 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 29 for EDD source code example and Figure 30 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 29 – EDD source code example for a layout for columns in stacked group

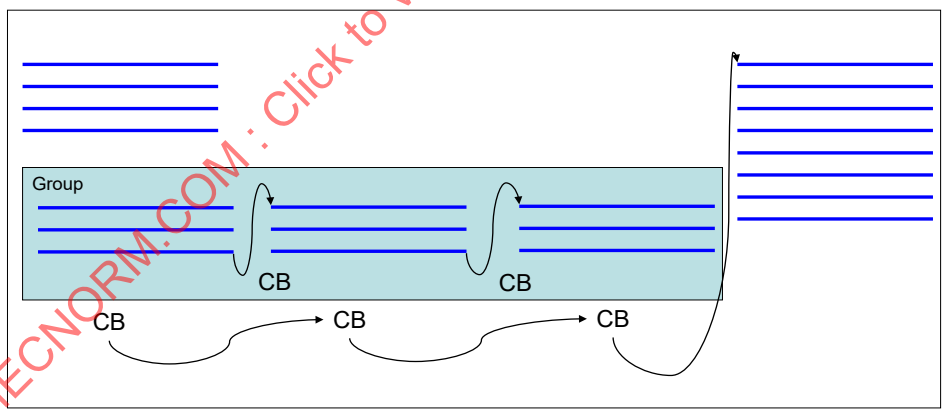


Figure 30 – Layout for columns in stacked group

Graphs inside groups shall be handled equivalently as multiple columns inside nested group, see Figure 31 for EDD source code example and Figure 32 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,
                GroupWithThreeElementsOneColumnsBreakChartWidthSmall,
                COLUMNBREAK,
                COLUMNBREAK,
                COLUMNBREAK,
                Element8,
                Element9,
                Element10,
                Element11,
                Element12,
                Element13,
                Element14,
                Element15
        }
}
```

Figure 31 – EDD source code for layout for columns with GRAPHs in stacked group example

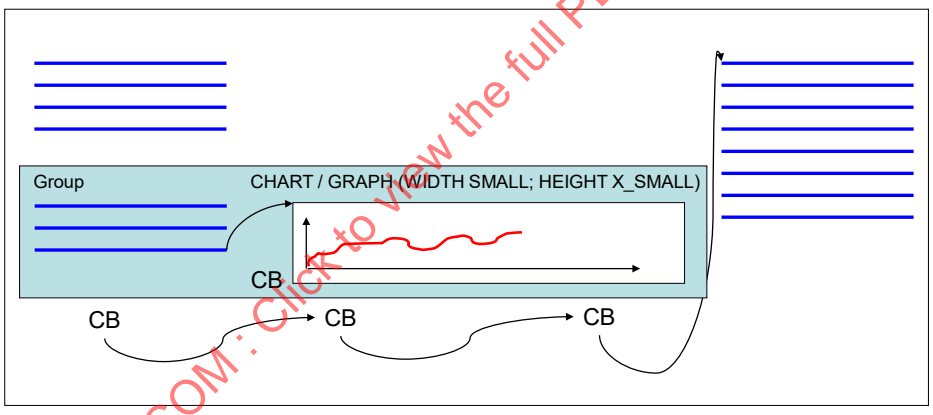


Figure 32 – Layout for columns with GRAPHs in stacked group

4.7.5 Layout examples

4.7.5.1 General

Layout examples in 4.7.5 illustrate layouts that would be rendered according to example source code. Layout examples that illustrate specific differences in optimized column width layout compared to equal column width layout are indicated in clauses marked with “optimized-column-width layout”.

4.7.5.2 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 33 – Example of an EDD for an overview menu

The EDD in Figure 33 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 34.

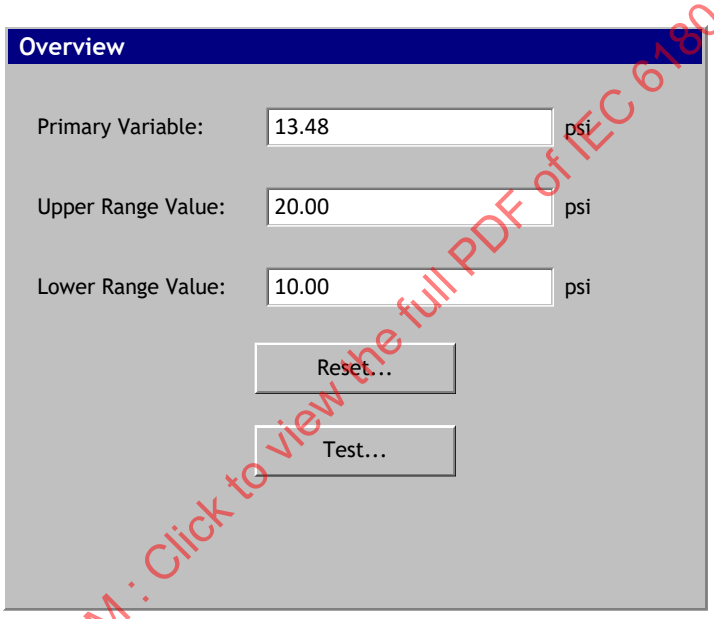


Figure 34 – Example of an EDD application for an overview window

4.7.5.3 Single-column layout (optimized-column-width layout)

In a single-column layout all menu items are arranged in one column. Figure 36 shows a single-column layout containing menu items with different width requirements. Figure 35 shows the corresponding source code.

```
MENU DIALOG_LayoutTest_SingleColumn
{
        LABEL           "Single Column Layout";
        STYLE           DIALOG;
        ITEMS
        {
                "Elapsed time since last check",
                var_DURATION,
                method_ResetTimer,
                var_ASCII,
                var_FLOAT,
                var_ENUM_Color
        }
}
```

Figure 35 – EDD source code for a layout with menu items spanning a single column

The width of the entire column is determined by totalizing the width of

- the widest label, which is "ENUMERATED",
- the widest menu item, which is "var_ENUM_Color" containing one extra wide enumeration element,
- the widest unit, which is "hh:mm:ss",
- the variable status, if provided.

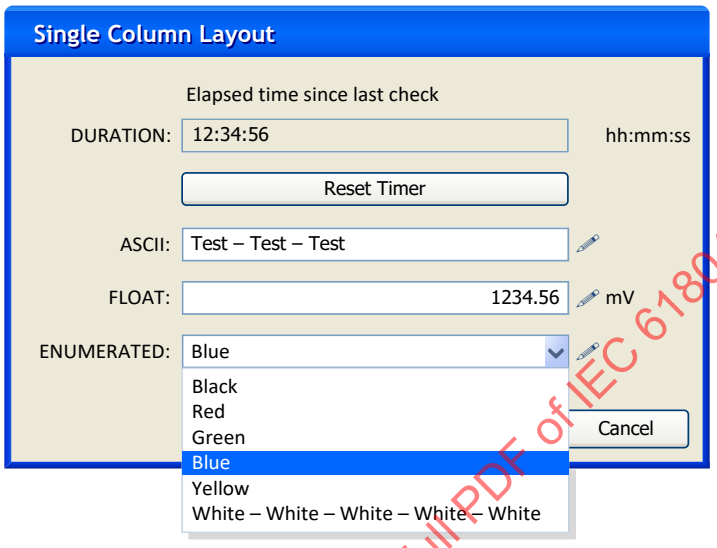


Figure 36 – Example of a layout with menu items spanning a single column

4.7.5.4 Multi-column and -row layout

Multiple columns of variables, methods, groups, images, graphs and charts may also be used, see Figure 37.

```
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 37 – Example of an EDD using COLUMNBREAK

A COLUMNBREAK in Figure 37 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 38.

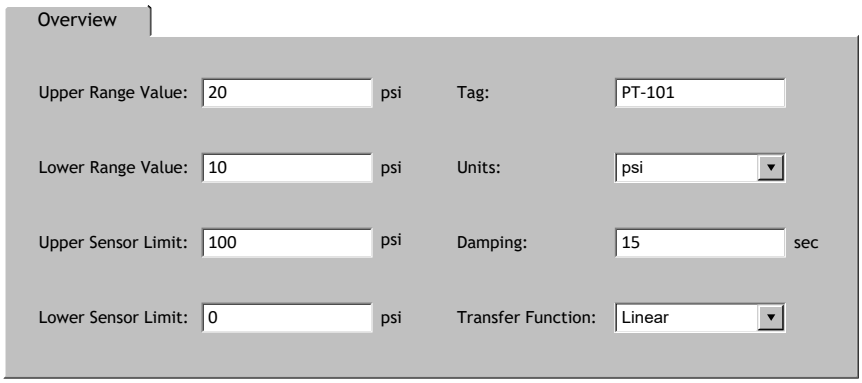


Figure 38 – Example of an EDD application for an overview window

A ROWBREAK in Figure 39 is used to place the following menu items beginning on the left side, see Figure 40.

```
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 39 – EDD example for an overview window

Figure 40 shows the result of Figure 39 in an EDD application. Blue dashed boxes show the available space.

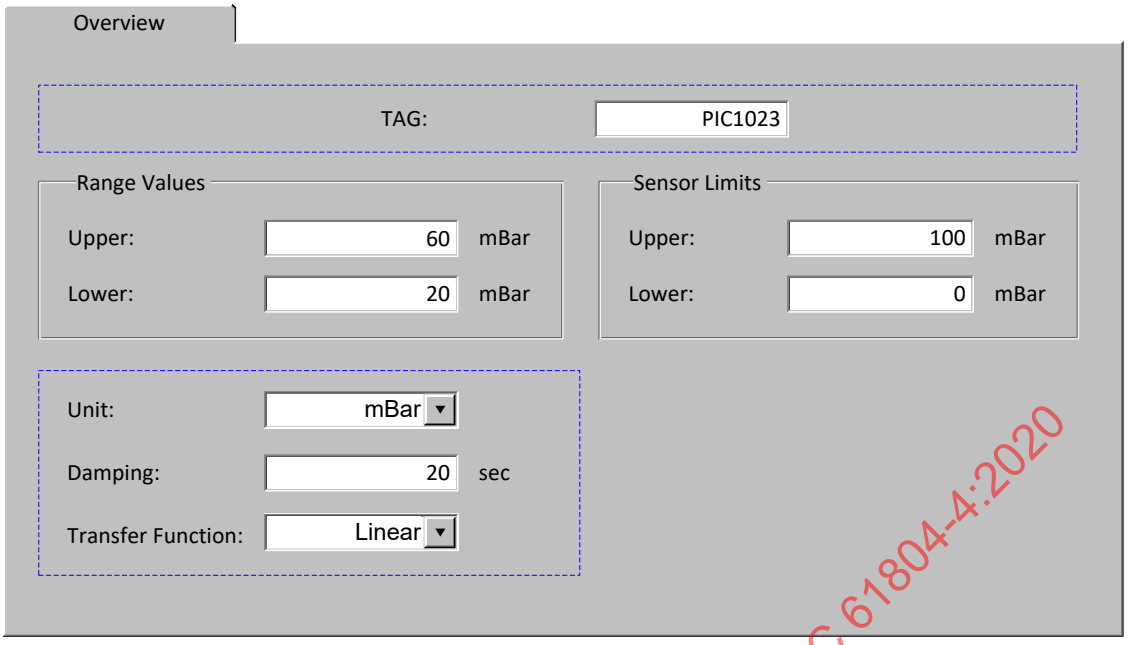


Figure 40 – Example of an EDD application for an overview window

4.7.5.5 Multi-column and -row layout (optimized-column-width layout)

In a multi-column layout the menu items are arranged in different columns depending on the number of COLUMNBREAKs within a row. The number of columns may vary from row to row.

Figure 42 shows a layout example containing three rows with a different number of columns. The 1st and the 3rd rows have 1 column, whereas the 2nd row has 2 columns with different widths. Figure 41 shows the corresponding source code.

```
MENU menu_Layout_With_ImageColumn
{
        LABEL   "Layout with 'small Image' Column";
        STYLE   DIALOG;
        ITEMS
        {
                "The quick brown fox jumps over the lazy dog. - "
                "The quick brown fox jumps over the lazy dog. - "
                "The quick brown fox jumps over the lazy dog.",
                ROWBREAK,
                var_DATE,
                var_ASCII,
                var_FLOAT,
                var_ENUM_COLOR,
                COLUMNBREAK,
                image_RED       (INLINE),
                image_YELLOW    (INLINE),
                image_GREEN     (INLINE),
                image_BLUE      (INLINE)
                ROWBREAK,
                method_ResetValues
        }
}
```

Figure 41 – EDD source code for a layout with small in-line images

The entire width is determined by the content of the 2nd row which has two columns. The width of the 1st column is determined by totalizing the width of

- the widest label, which is "ENUMERATED",
- the widest menu item, which is "var_ENUM_COLOR" containing one extra wide enumeration element,
- the widest unit, which is "mV",
- the variable status.

The width of the 2nd column is determined by the small in-line images.

The content of the 1st row (constant string) and the 3rd row (button) is aligned and expanded to span the same width like the 2nd row. Thus the input field of the 1st and the 3rd row are wider than the respective maximum widths (see Table 23).

Due to the flexible layout structure provided by the optimized-column-width layout, it's possible to place the small in-line images close to the 1st column, e.g., to emphasize a special relation between these columns.

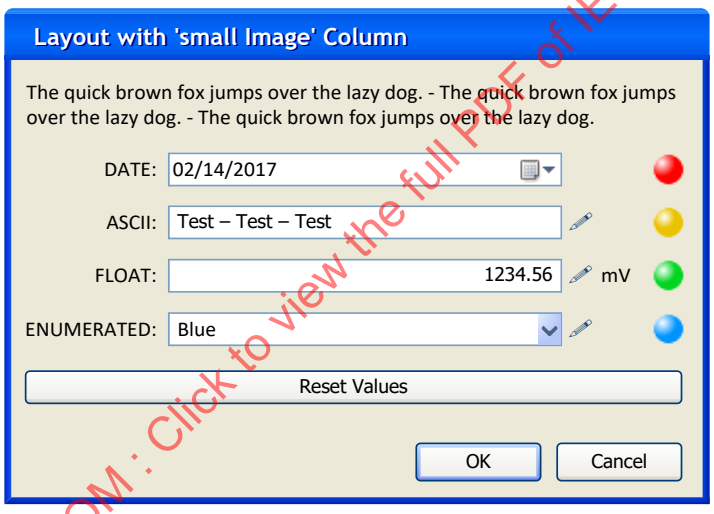


Figure 42 – Example of a layout with small in-line images

Figure 44 shows a layout example containing 3 rows with a different number of columns. The 1st row contains 3 columns, the 2nd row contains a nested group with 2 columns, and the 3rd row contains variables spanning a different number of columns by defining the WIDTH attribute. Figure 43 shows the corresponding source code.

```
MENU menu_Layout_MultiColumn_With_Group

        LABEL   "Multi-column Layout";
        STYLE   DIALOG;
        ITEMS
        {
                var_INTEGER,
                COLUMNBREAK,
                var_DOUBLE,
                COLUMNBREAK,
                var_DATE,
                ROWBREAK,
                menu_Group_2C,
                ROWBREAK,
                var_ASCII,                      // ASCII variable with no WIDTH → default
                var_ASCII_Width_SMALL,          // ASCII variable with 'WIDTH SMALL'
                var_ASCII_Width_MEDIUM          // ASCII variable with 'WIDTH MEDIUM'
        }
}

MENU menu_Group_2C
{
        LABEL   "Group - 2C";                   // GROUP with 2 columns
        STYLE   GROUP;
        ITEMS
        {
                var_ASCII,
                var_ENUMERATED,
                COLUMNBREAK,
                var_FLOAT,
                var_INTEGER
        }
}
```

Figure 43 – EDD source code for a multi-column layout with GROUP

The 1st row has 3 columns by COLUMNBREAKs and the items in the 3rd row span up to 3 columns by WIDTH. Thus the items in these rows are aligned.

The width of the 1st column is determined by totalizing the width of

- the widest label, which is "ASCII (MEDIUM = 3C)" in the 3rd row,
- the widest input field (value), which is "var_ASCII" in the 3rd row,
- the variable status.

The width of the 2st column is determined by totalizing the width of

- the label "DOUBLE" in the 1st row,
- the input field (value), which is "var_DOUBLE" in the 1st row,
- the variable status,
- the unit "L/h".

The width of the 3st column is determined by totalizing the width of

- the label "DATE" in the 1st row,
- the input field (value), which is "var_DATE" in the 1rd row.

The resulting width of the layout is determined by totalizing the width of the 1st, the 2nd and the 3rd column. Variable "var_ASCII_Width_SMALL" in the 3rd row is expanded to span 2 columns; variable "var_ASCII_Width_MEDIUM" is expanded to span 3 columns and aligned. The respective input fields are aligned, which is emphasized by the auxiliary lines in Figure 44.

Due to the ROWBREAKs, the content of the GROUP in the 2nd is initially calculated independently with 2 columns. Then the 2 columns of the GROUP are expanded to span the same width like the other rows.

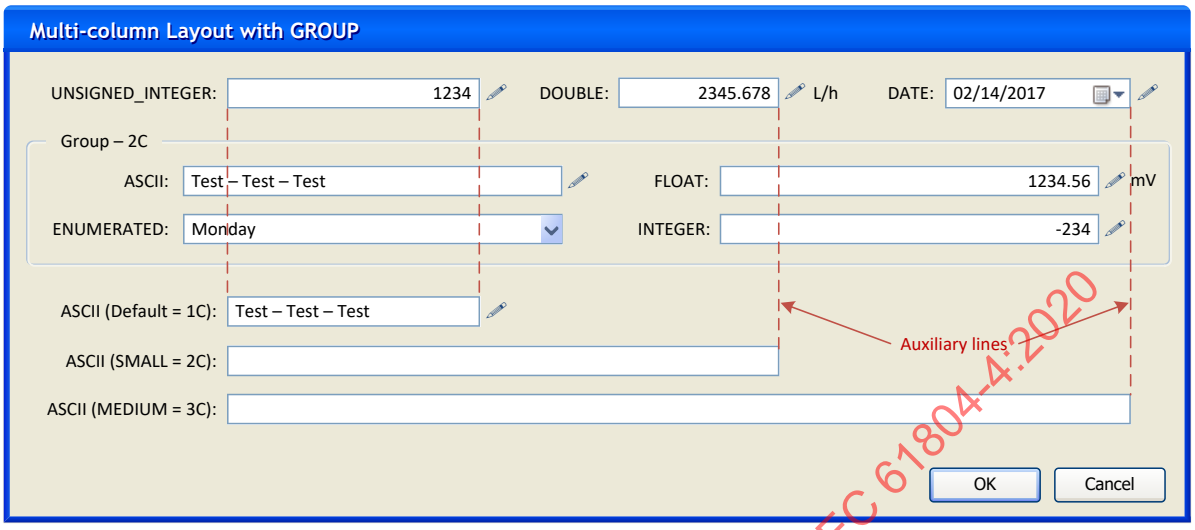


Figure 44 – Example of a multi-column layout with GROUP

4.7.5.6 In-line graphs and charts

Graphs and charts may appear within the columns just like variables and methods, see Figure 45 and Figure 46.

```
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 45 – Example of an EDD for in-line graphs and charts

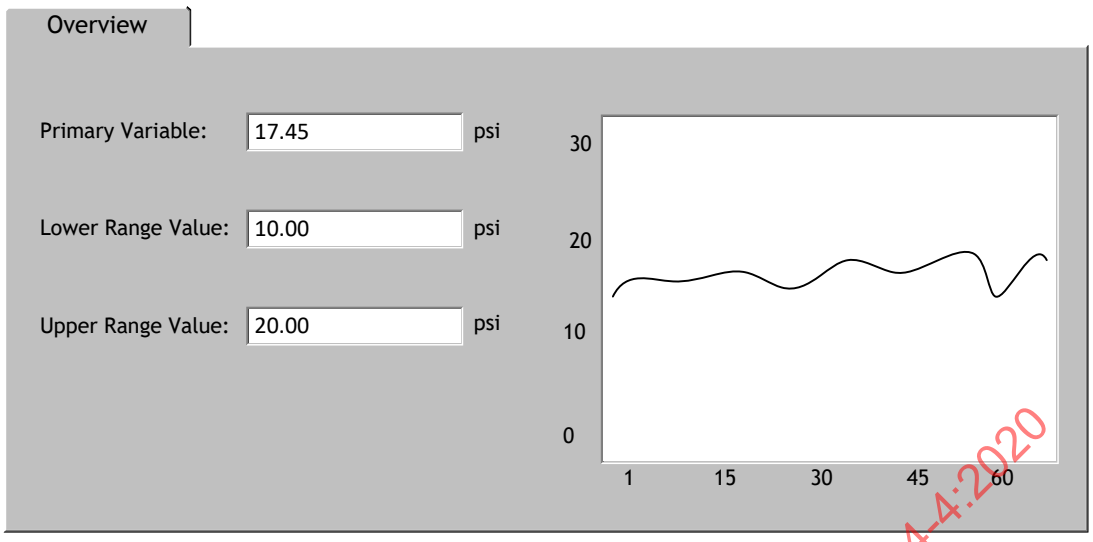


Figure 46 – Example of an EDD application for an in-line graph

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 4.7.5.7 shall apply.

4.7.5.7 Full-width graphs and charts

The following example illustrates the layout of graphs and charts spanning multiple columns for both layout types. The related EDD source code is shown in Figure 47.

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

MENU overview_page
{
    LABEL "Overview";
    STYLE PAGE;
    ITEMS
    {
        primary_variable,           /* variable */
        COLUMNBREAK,                /* column break */
        lower_range_value,          /* variable */
        COLUMNBREAK,                /* column break */
        upper_range_value,          /* variable */
        pv_graph,                   /* graph with WIDTH X_LARGE, occupies 5 columns */
        ROWBREAK,                   /* row break */
        reload_pv_graph,            /* method */
        COLUMNBREAK,                /* column break */
        save_pv_graph               /* method */
    }
}
```

Figure 47 – Example of an EDD for full-width graphs and charts

Equal-column-width layout:

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.

Figure 48 shows the resulting equal-column-width layout.

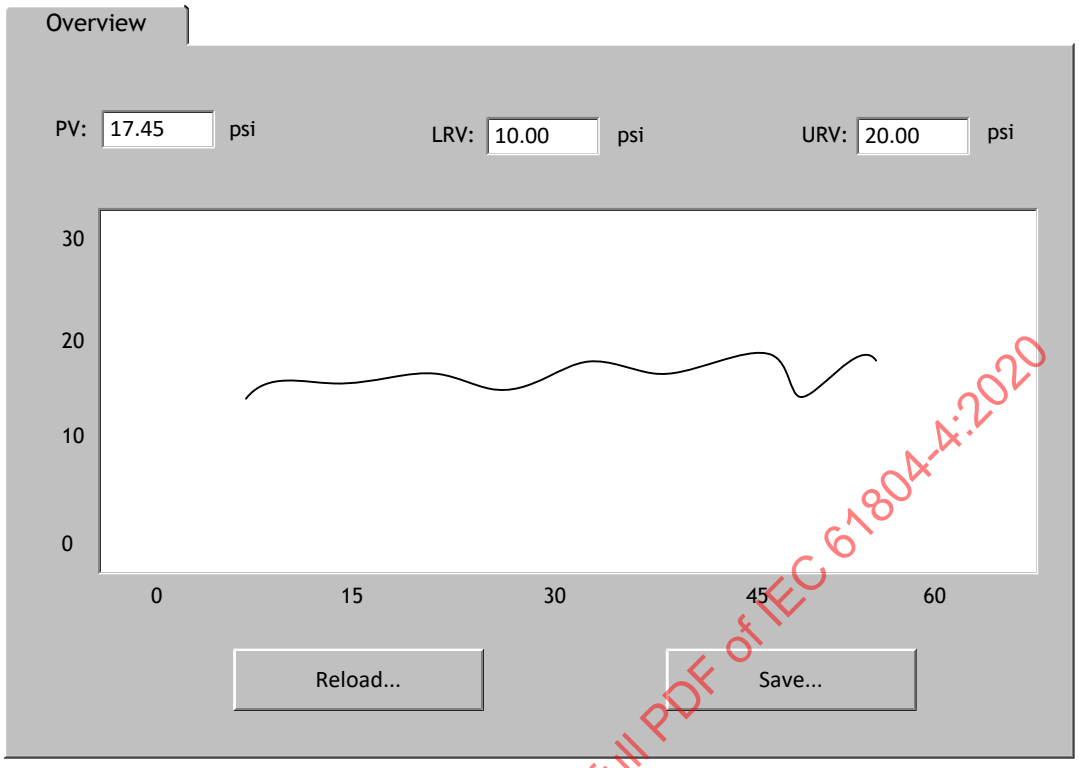


Figure 48 – Example of an EDD application for a full-width graph in a layout with equal column width

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

Optimized-column-width layout:

According to the EDD source code is shown in Figure 47 the 1st row contains 3 columns, the 1st and the 2nd column with one menu item each and the 3rd column with 2 menu items. Since columns may span to a column higher than the 5th, the graph is rendered inline from the 3rd column without being moved to the next row through an implied ROWBREAK. Figure 49 shows the resulting optimized-column-width layout.

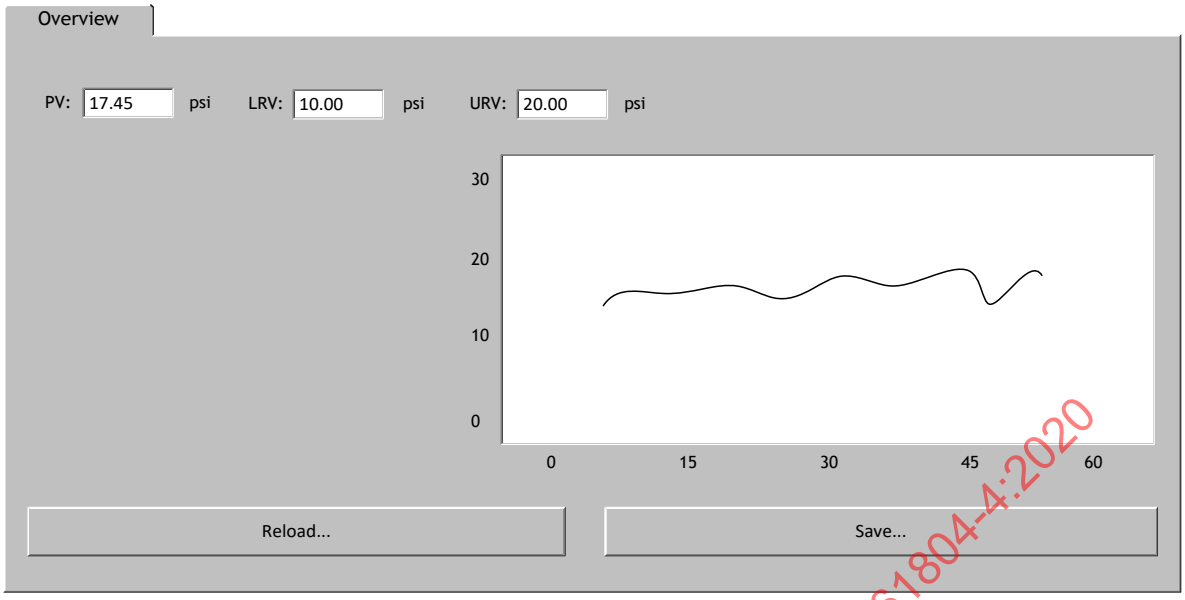


Figure 49 – Example of an EDD application for a full-width graph in a layout with optimized column width

If the graph should be rendered from the 1st column, the EDD developer shall insert a ROWBREAK in before the graph (pv_graph).

4.7.5.8 Nested containers

Containers can be nested within other containers. Figure 50 and Figure 51 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 50 – Example of an EDD for nested containers

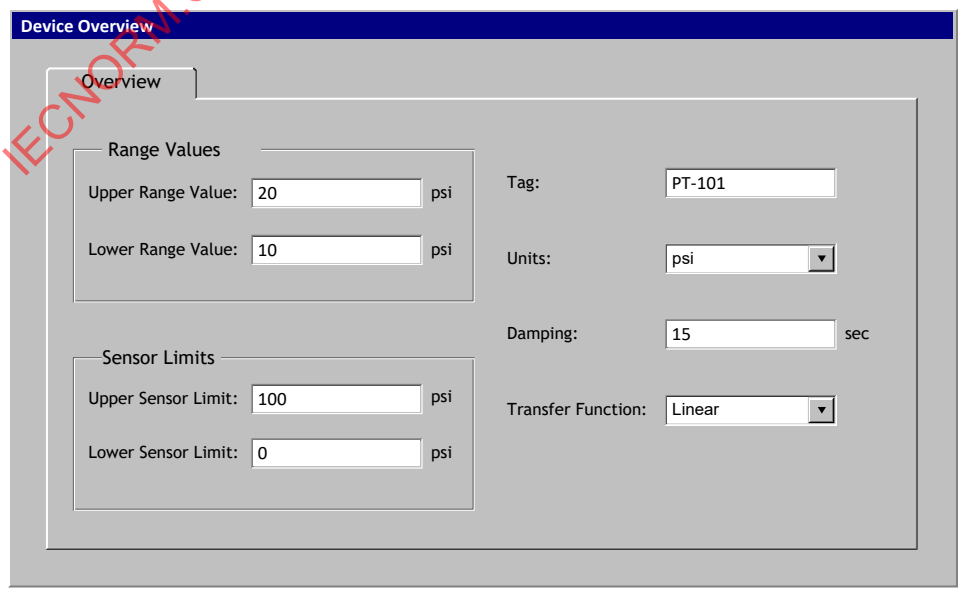


Figure 51 – Example of an EDD application for nested containers

4.7.5.9 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 52 and Figure 53. 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 52 – Example of an EDD for EDIT_DISPLAYS

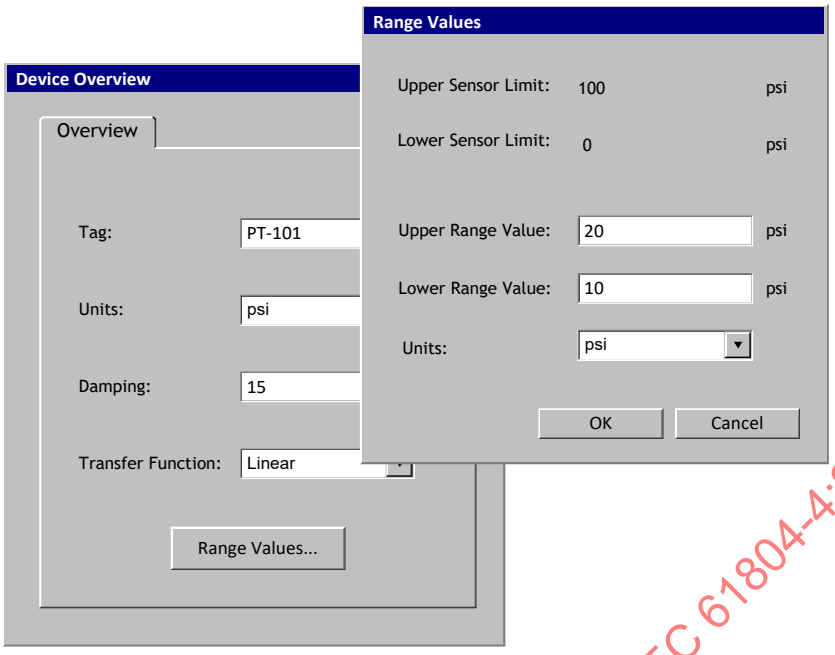


Figure 53 – Example of an EDD application for EDIT_DISPLAYS

4.7.5.10 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 54 and Figure 55. Otherwise, the image is displayed across the width of the container.

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

Figure 54 – Example of an EDD for images

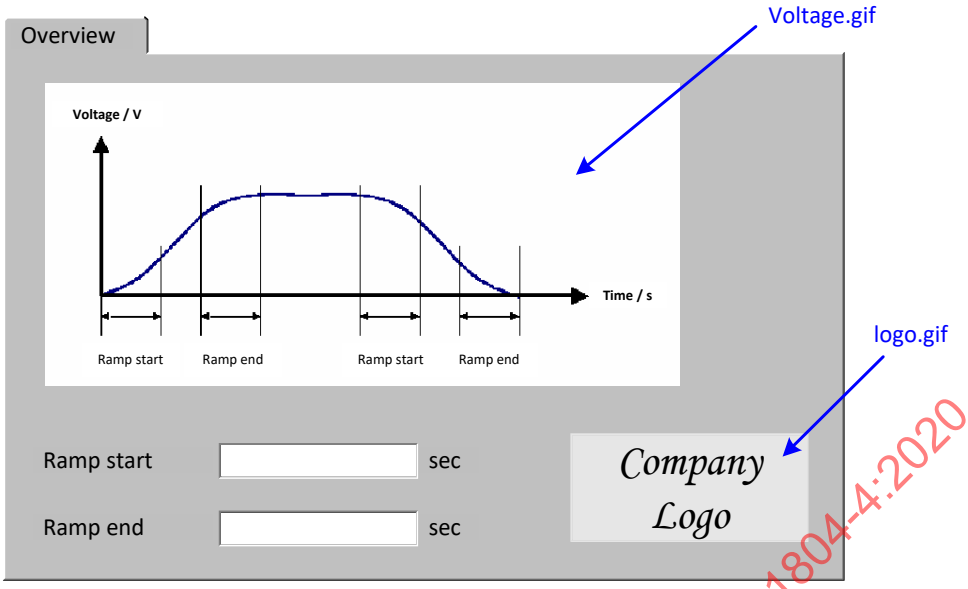


Figure 55 – Example of an EDD application for images

Optimized-column-width layout:

In a layout with optimized column width, the image shall not be scaled down, even if INLINE is specified. Figure 57 shows a layout example with a large image rendered inline. Figure 56 shows the corresponding EDD source code.

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

Figure 56 – Example of an EDD for large inline-images

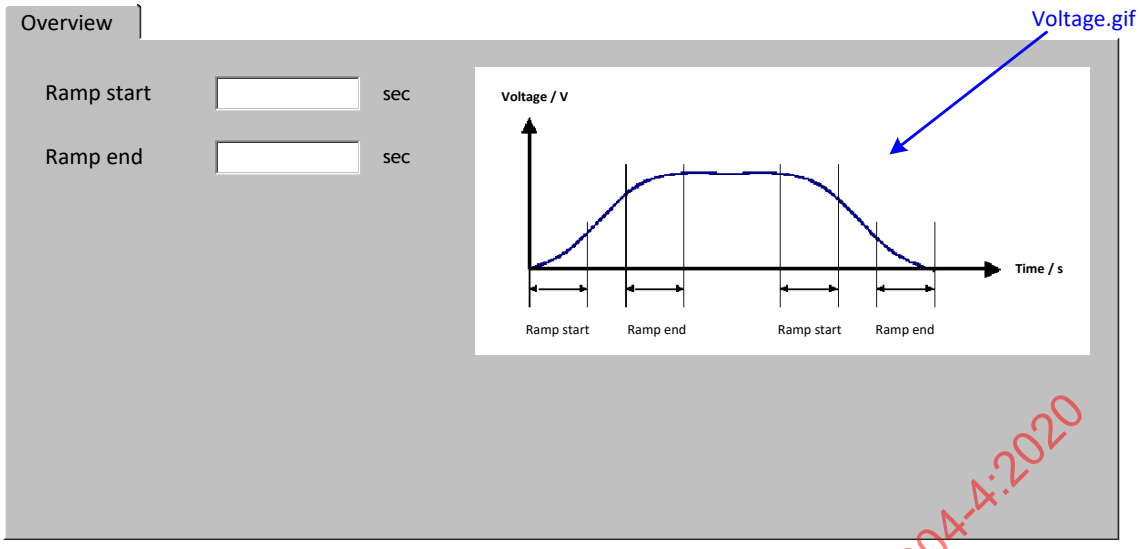


Figure 57 – Example of layout with a large inline-image

4.7.6 Conditional user interface

4.7.6.1 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.

4.7.6.2 VISIBILITY

The attribute VISIBILITY of the basic constructs (e.g. MENU, VARIABLE, IMAGE, GRAPH, CHART, GRID) allows to control 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.

4.7.6.3 Conditional expressions in MENU ITEM

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

4.7.6.4 VALIDITY

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

For both online and offline sessions, if a construct VALIDITY is evaluated from TRUE to FALSE, EDD application shall remove the construct from being displayed.

For both online and offline sessions, if a construct VALIDITY is evaluated from FALSE to TRUE, EDD application shall display the construct.

For online session, when a VARIABLE VALIDITY is evaluated from FALSE to TRUE, EDD application shall display the VARIABLE value as uninitialized (see Table 20), until the VARIABLE is read from the device or modified during the same edit session.

For offline session, when a VARIABLE VALIDITY is evaluated from FALSE to TRUE, EDD application shall display the VARIABLE value from the current cache.

VARIABLES with VALIDITY FALSE referenced from a MENU should not be read from the device.

VALIDITY does affect the screen layout. In the screen layout, no space is reserved if the element is not valid. Data items in a GRID that have VALIDITY = FALSE shall behave the same as if VISIBILITY=FALSE (i.e. the space is reserved, but the cell is empty).

Figure 58 is an EDD example for VALIDITY in online session. Table 25, Table 26, Table 27 and Table 28 show how an EDD application works with VALIDITY in online session

```
VARIABLE option
{
        LABEL "Option";
        HANDLING READ & WRITE;
        TYPE ENUMERATED
        {
                {0, "Disable"},
                {1, "Internal Option"},
                {2, "External Option"}
        }
}

VARIABLE internal
{
        LABEL "Internal";
        VALIDITY IF (option == 1) {TRUE;} ELSE {FALSE;}
        HANDLING READ & WRITE;
        TYPE FLOAT
        {
                DISPLAY_FORMAT ".1f";
                EDIT_FORMAT "8.1f";
        }
}

VARIABLE external
{
        LABEL "External";
        VALIDITY IF (option == 2) {TRUE;} ELSE {FALSE;}
        HANDLING READ & WRITE;
        TYPE FLOAT
        {
                DISPLAY_FORMAT ".1f";
                EDIT_FORMAT "8.1f";
        }
}

MENU selection_option
{
        LABEL "Selection";
        STYLE DIALOG;
        ITEMS
        {
                option (DISPLAY_VALUE),
                internal (DISPLAY_VALUE),
                external (DISPLAY_VALUE)
        }
}
```


Figure 58 – EDD example for VALIDITY in online session

Table 25 – Example 1 VALIDITY in an online session

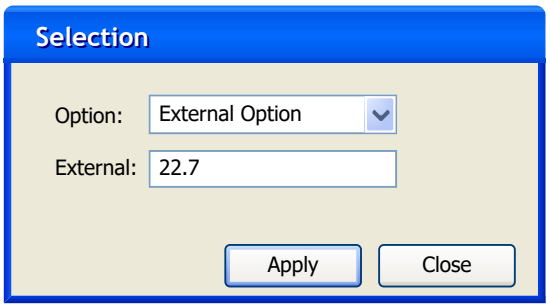
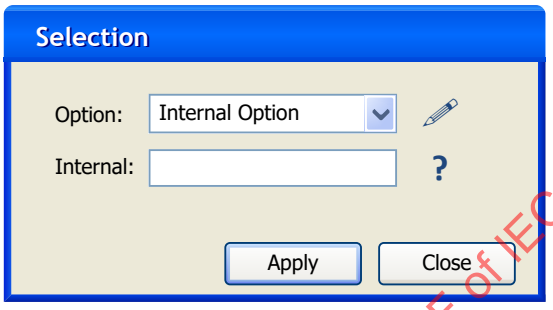
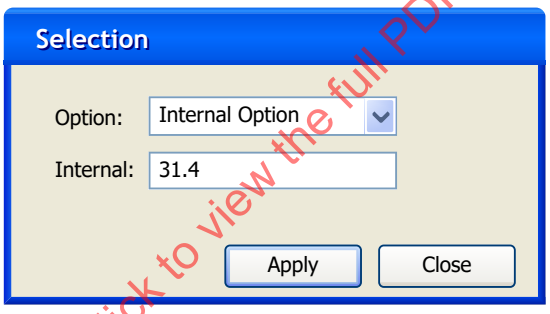
Step	User interface	Description
Step 1: Initial state “Option” = “External Option”		“Internal” is VALIDITY FALSE and not displayed. “External” is VALIDITY TRUE and is displayed.
Step 2: Edit “Option” = “Internal Option”		“External” becomes VALIDITY FALSE. “External” is removed from display. “Internal” becomes VALIDITY TRUE. “Internal” is displayed editable and blank with indication of “uncertain” state close to the value field.
Step 3: Press “Apply” to commit the changed value		“Option” is committed “Internal” is not committed. “Internal” is read and displayed.

Table 26 – Example 2 VALIDITY in an online session

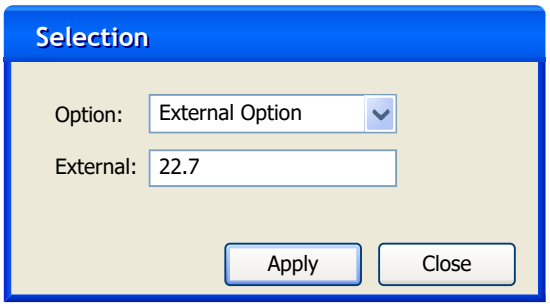
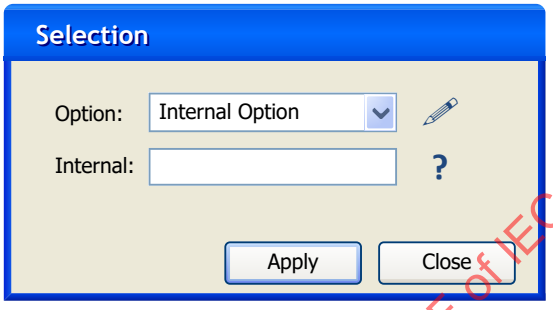
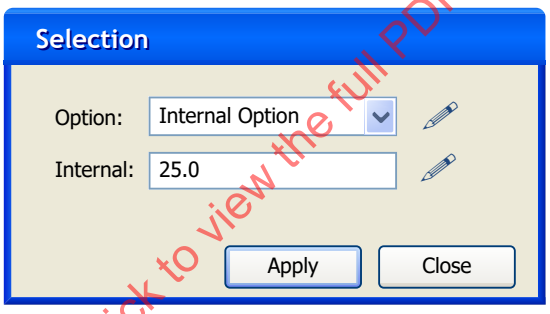
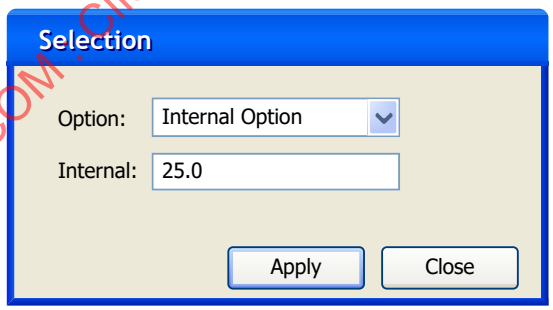
Step	User interface	Description
Step 1: Initial state “Option” = “External Option”		“Internal” is VALIDITY FALSE and not displayed. “External” is VALIDITY TRUE and is displayed.
Step 2: Edit 2 “Option” = “Internal Option”		“External” becomes VALIDITY FALSE. “External” is removed from display. “Internal” becomes VALIDITY TRUE. “Internal” is displayed editable and blank with indication of “uncertain” state close to the value field.
Step 3: Edit “Internal” to 25.0		Edited value is displayed with indication of “changed” state close to the value field.
Step 4: Press “Apply” to commit the changed value		“Option” is committed. “Internal” is committed.

Table 27 – Example 3 VALIDITY in an online session

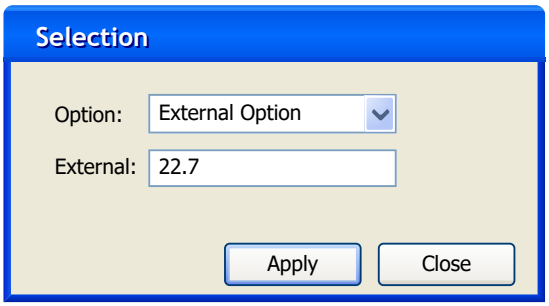
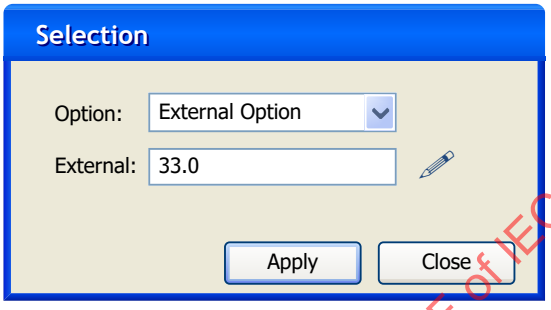
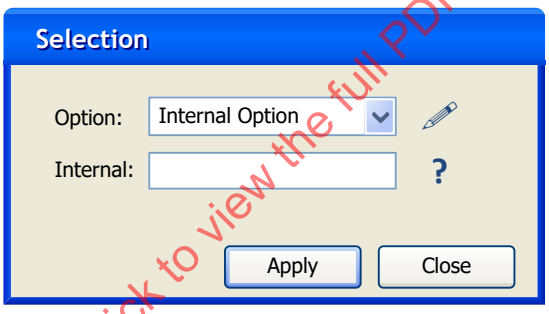
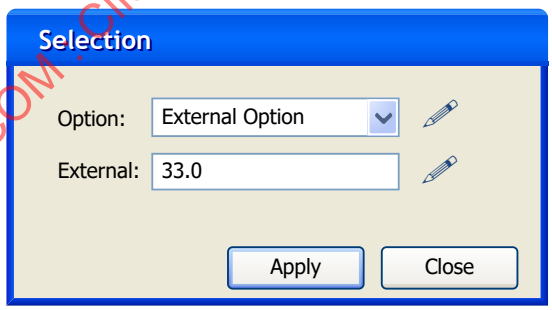
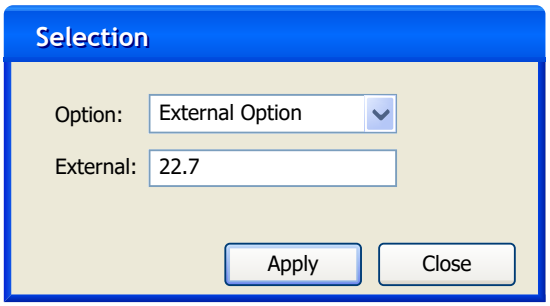
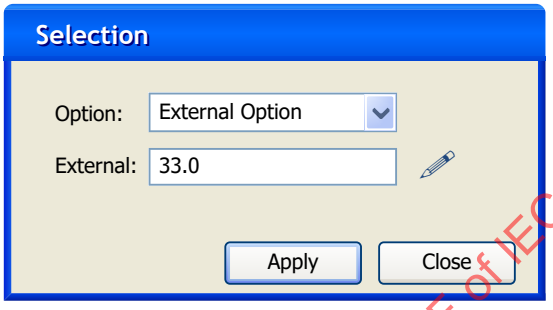
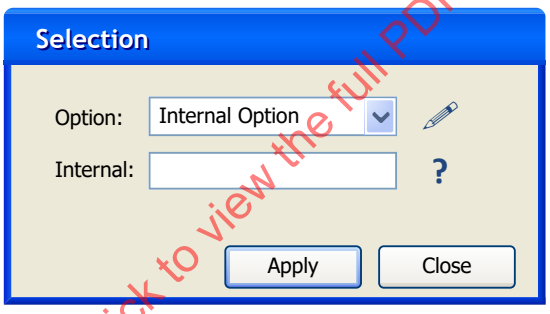
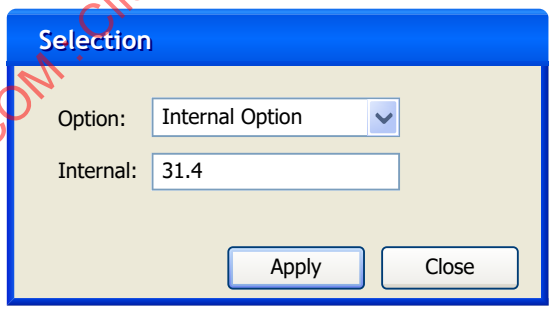
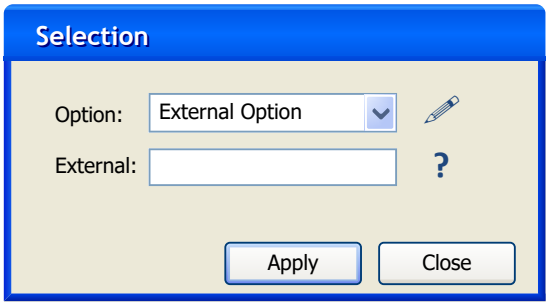
Step	User interface	Description
Step 1: Initial state “Option” = “External Option”		“Internal” is VALIDITY FALSE and not displayed. “External” is VALIDITY TRUE and is displayed.
Step 2: Edit “External” to 33.0		Edited value is displayed with indication of “changed” state close to the value field.
Step 3: Make “External” VALIDITY FALSE by editing “Option” to “Internal Option”		“External” becomes VALIDITY FALSE. “External” is removed from display.
Step 4: Make “External” VALIDITY TRUE again by editing “Option” to “External Option”		“External” becomes VALIDITY TRUE. “External” is displayed with previous edited value with indication of “changed” state close to the value field.

Table 28 – Example 4 VALIDITY in an online session

Step	User interface	Description
Step 1: Initial state “Option” = “External Option”		“Internal” is VALIDITY FALSE and not displayed. “External” is VALIDITY TRUE and is displayed.
Step 2: Edit “External” to 33.0		Edited value is displayed with indication of “changed” state close to the value field.
Step 3: Make “External” VALIDITY FALSE by editing “Option” to “Internal Option”		“External” becomes VALIDITY FALSE. “External” is removed from display.
Step 4: Press “Apply” to commit the changed value		“Option” is committed. “Internal” is not committed. “Internal” is read and displayed. Previously edited but VALIDITY FALSE “External” is not committed.
Step 5: Make “External” VALIDITY TRUE again by editing “Option” to “External Option”		“External” becomes VALIDITY TRUE. “External” is displayed editable and blank with indication of “uncertain” state close to the value field.

4.8 Graphical elements

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. Colored regions (bands) can be created using limit lines (i.e., line types LOW_LOW_LIMIT, LOW_LIMIT, HIGH_LIMIT, and HIGH_HIGH_LIMIT).

For limit lines of the same line type sharing the same y-axis, the limit lines shall be sorted by value. The resulting ordered list of limit lines shall create successive regions (bands) on meter type CHARTs. For low limit types, the upper boundary of a limit region shall be set by the value of the corresponding limit line. For high limit types, the lower boundary of a limit region shall be set by the value of the corresponding limit line. The color of a limit region shall be the LINE_COLOR of the corresponding limit line. The lower boundary of the first limit region shall start with the axis minimum. The upper boundary of the final limit region shall continue to the axis maximum.

Figure 59 shows an example of limit regions rendered by an application given the EDD definition shown in Figure 60.

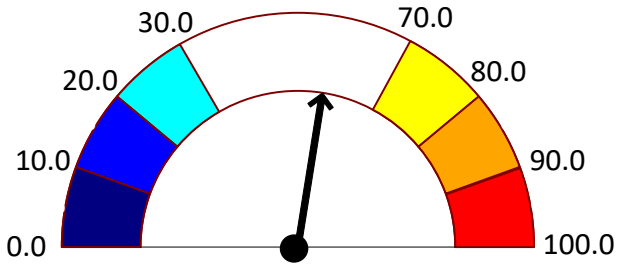


Figure 59 – Example of an EDD application for a gauge with limit regions

```
VARIABLE local_minlow_limit
{
   CLASS LOCAL;
   TYPE FLOAT;
   DEFAULT_VALUE 10.0;
}

VARIABLE local_midlow_limit
{
   CLASS LOCAL;
   TYPE FLOAT;
   DEFAULT_VALUE 20.0;
}

VARIABLE local_maxlow_limit
{
   CLASS LOCAL;
   TYPE FLOAT;
   DEFAULT_VALUE 30.0;
}

VARIABLE local_minhi_limit
{
   CLASS LOCAL;
   TYPE FLOAT;
   DEFAULT_VALUE 70.0;
}

VARIABLE local_midhi_limit
{
   CLASS LOCAL;
   TYPE FLOAT;
   DEFAULT_VALUE 80.0;
}

VARIABLE local_maxhi_limit
{
   CLASS LOCAL;
   TYPE FLOAT;
   DEFAULT_VALUE 90.0;
}

AXIS gaugeAxis
{
   MIN_VALUE deviceVariables[0].LOWER_SENSOR_LIMIT;  //  0.0 For this example
   MAX_VALUE deviceVariables[0].UPPER_SENSOR_LIMIT;  //100.0 For this example
}

SOURCE minlowSource
{
   LINE_TYPE LOW_LIMIT;
   LINE_COLOR NAVY;
   Y_AXIS gaugeAxis;
   MEMBERS
   {
      LIMIT, local_minlow_limit;
   }
}

SOURCE midlowSource
{
   LINE_TYPE LOW_LIMIT;
   LINE_COLOR BLUE;
   Y_AXIS gaugeAxis;
   MEMBERS
   {
      LIMIT, local_midlow_limit;
   }
}

SOURCE maxlowSource
{
   LINE_TYPE LOW_LIMIT;
   LINE_COLOR AQUA;
   Y_AXIS gaugeAxis;
   MEMBERS
   {
      LIMIT, local_maxlow_limit;
```

```
    }
}

SOURCE minhiSource
{
   LINE_TYPE HIGH_LIMIT;
   LINE_COLOR YELLOW;
   Y_AXIS gaugeAxis;
   MEMBERS
   {
      LIMIT, local_minhi_limit;
   }
}

SOURCE midhiSource
{
   LINE_TYPE HIGH_LIMIT;
   LINE_COLOR ORANGE;
   Y_AXIS gaugeAxis;
   MEMBERS
   {
      LIMIT, local_midhi_limit;
   }
}

SOURCE maxhiSource
{
   LINE_TYPE HIGH_LIMIT;
   LINE_COLOR RED;
   Y_AXIS gaugeAxis;
   MEMBERS
   {
      LIMIT, local_maxhi_limit;
   }
}

CHART exampleGauge
{
   LABEL "Example Gauge";
   TYPE GAUGE;
   MEMBERS
   {
      GAUGE_VALUE,  valueSource;
      MINLOW_LIMIT, minlowSource;
      MIDLOW_LIMIT, midlowSource;
      MAXLOW_LIMIT, maxlowSource;
      MINHI_LIMIT,  minhiSource;
      MIDHI_LIMIT,  midhiSource;
      MAXHI_LIMIT,  maxhiSource;
   }
}
```

Figure 60 – Example of an EDD for a gauge with limit regions

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).

5 EDDL data description

5.1 EDDL application stored device data

5.1.1 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.

5.1.2 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.

NOTE Invalid FILE members do not invalidate the entire FILE. However, EDD applications will recognize when a member or a portion of a member is invalid. For example: An EDD application could simply take a snapshot of the MEMBERS and identify the invalid member. Subsequently, that EDD Item can be initialized to invalid the next time FILE for this device instance is loaded by the EDD application.

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 61 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 61 – 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 62 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 62 – Example of comparing valve signatures

In the example shown in Figure 62, 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.

5.1.3 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 63. 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 63 – Example of more complex file declaration

In the example of Figure 63, 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 64 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 64 – 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 an index info variable. 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 65 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,     ping_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 65 – Example of an EDD that inserts, replaces, or compares radar signals

5.2 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.

5.3 Initialization of EDD instances

5.3.1 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.

5.3.2 Initialization support

If the user selects a device for the first time, then a new instance is created by the EDD application with the related EDD. The EDD variable values for newly created instances shall be initialized using the following rules.

a) Variable, Value Array, and List initialization using EDDL COMPONENTs if EDDL COMPONENTs exist, else

b) Variable initialization with EDDL INITIAL_VALUEs if INITIAL_VALUEs exist, else

c) Variable initialization with DEFAULT_VALUEs if DEFAULT_VALUEs exist, else

d) Variables initialized with the variable type initial value.

In addition, the user may select a TEMPLATE that contains further initializations.

5.3.3 TEMPLATE

TEMPLATEs specify default parameter values for different uses of a device, i.e., they are application specific. The EDD application shall provide a list of all TEMPLATEs defined in the EDD to the user. If the user selects 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.

TEMPLATE initialization of the VARIABLE that is employed in the TYPE attribute of an ARRAY or LIST shall not affect the ARRAY or LIST elements.

5.4 Device model mapping

5.4.1 BLOCK_A

A FOUNDATION fieldbus and ISA100_Wireless 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 in Figure 66 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_variables_root_menu_ai;   /* block based menu */
   }
}

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

MENU process_variables_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 66 – Example of a BLOCK_A

5.4.2 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.

6 EDDL METHOD programming and usage of builtins

6.1 Method environment

6.1.1 General

All of the lexical conventions specified in IEC 61804-3 apply to methods and shall be used when developing the ANSI C based DEFINITION of the procedure specified by the METHOD. The translation has been performed from the EDD source including execution of the preprocessor commands prior to the host application accessing or using the EDD.

6.1.2 Security

Methods are executed in a secure "sandbox" enclosed within the EDD application. All method activity occurs within this sandbox. EDD applications shall implement only the Standard Library Builtin functions. These library functions cannot launch EDD application-native programs or access an external Dynamic Link Library (DLL). Direct access to the system's local disk shall not be allowed. The focused and limited scope of the standard library functions ensures secure use of EDDs and controls access to system resources from within the EDD methods. The EDD application is responsible for ensuring its security and the security of the system it operates on.

Conceptually, this sandbox is created when a method is invoked or triggered. The sandbox exists until the method completes normally or via abort method processing. All of the method subroutines and abort methods will be executed in the sandbox created upon method startup.

6.1.3 Device data

All device data such as VARIABLES and ARRAYS appear as global variables to the C-code defining the procedure. VARIABLES can be directly accessed like any variable declared in the ANSI C-language.

Normally the primary, device data cache is being updated continuously by the EDD application to ensure it has an up-to-date copy of the data contained in the field device. When the sandbox is created, the method's (secondary) data cache is initialized from the primary cache. If the primary cache is not fully initialized, then commands will be dispatched as needed for data items referenced in the method. For security purposes, the method operates on this cached set of values rather then directly manipulating the primary cache. This allows, for example, a method to abort without adversely affecting the field device's configuration.

Methods modify data in its (secondary) cache. The method shall not abort if invalid data is accessed (in any fashion) by the method. Sending commands writes-through-to and reads-back-from the field device via the primary cache. This ensures the primary cache remains synchronized with the data values in the field device. Caching for methods is specified in 8.2.5.

When a method is operating it has full control of the EDD application. Normally, no commands are sent and no data is updated unless the method sends the commands necessary to do so. This ensures that the field device can be placed in any special operating modes needed to perform the Standard Operation Procedure without generating errors or alarms. In some devices, dispatching an invalid command during, for example, calibration could cause errors to occur.

6.1.4 Method TYPE and parameters

Each method is a single function or procedure similar to ANSI C-functions. The ANSI-C standard library is not supported in the METHOD definition language, instead there is a library of EDDL Builtins available for use within METHODs (see IEC 61804-5 EDDL Library). METHODs are also allowed to call other METHODs (referred to as method-functions, see IEC 61804-5 EDDL Builtin Library). The parameter list (if there are any parameters) for the method-function is indicated in parenthesis immediately following the name of the METHOD. The parameters are comma separated list with each parameter listed along with the parameter's identifier. These identifiers have a scope limited to the METHOD's DEFINITION only.

Communication profiles: HART

Methods requiring parameters shall only be invoked from a METHOD DEFINITION.

These parameters become part of the METHOD'S environment and are accessed using their identifier the same as other ANSI C variables. In addition to the usual ANSI C variables, two EDDL specific parameter types shall be supported:

a) DD_STRING is a special string object encapsulating all strings and allowing strings to be manipulated. The DD_STRING type is similar to the ANSI C++ string template class. All of the Standard Builtin Library string function use DD_STRING.

b) DD_ITEM is a type references a EDD Item (e.g., a VARIABLE). This allows EDD Items to be passed into a METHOD-function as if it was ANSI C-variable.

Both DD_STRING and DD_ITEM shall always be passed by reference. ANSI C-variables (e.g., int, long float) can be passed by reference or by value.

Finally, the optional TYPE attribute allows METHOD-functions to return a value. The method return TYPE can be any of the ANSI C "basic types" or a DD_STRING may be returned.

6.1.5 Abort processing

A method can be aborted (abnormally terminated). The trigger for the abort can be the user manually aborting the method while execution is stopped waiting for user input, the method's definition programmatically aborting the method, or a system event (e.g., loss of communications with the field device).

Since the method is normally manipulating the device's, control system's or even the plant's state during the course of executing a Standard Operation Procedure, the method's writer shall be conscious of the ramifications of an abort occurring. If one does occur, the method's writer needs to be prepared to correct any conditions adverse to the plant, control system or field device.

To accomplish the necessary corrections, the method writer normally develops a series of abort methods. One abort method is usually developed for each condition that might be faced should an abort occur. The method defining the Standard Operation Procedure then designates the abort service method required. These may be deleted, added or changed during Standard Operation Procedure execution (see "Abnormal Termination Builtins" in IEC 61804-5).

A wide variety of abort trigger sources are available and can be selected depending on the needs of the Standard Operation Procedure being defined. These sources can be masked to disable the trigger or un-masked to allow an abnormal event to trigger the abort automatically. Retries can be performed automatically in some circumstances. All of these are under the control of the method writer (see "Communication Builtins" IEC 61804-5).

When an abort occurs, the list of abort functions is executed. Abort functions are still executing within the sandbox with the cached device data. Consequently, any changes made within the Standard Operation Procedure's method have not been committed to the device unless the appropriate command were sent prior to the abort occurring. The abort method is responsible for performing any actions necessary to return the device (and system) to proper operation. Commands may be sent and METHOD-functions may be called.

A METHOD-function can specify its own abort functions. Should an abort occur during the execution of the METHOD-function, its abort methods are called first, followed by the calling method's abort methods.

6.2 Implementation requirements

Method execution implementations shall follow these rules:

1) In a METHOD, when the independent VARIABLE in a REFRESH or UNIT relation is modified and sent to the device, the EDD application shall perform the refresh prior to using or displaying any of the dependent VARIABLES in the REFRESH or UNIT relation.
2) Any actions defined for VARIABLES shall be exercised (as appropriate) when accessed within a METHOD. For example, if a VARIABLE has a POST_EDIT_ACTION, then that action should be run after the VARIABLE is modified using the GET_DEV_VAR_VALUE function.
3) If a VARIABLE is modified and then any METHOD is run, the METHOD will have access to the modified value, even if that value hasn't yet been sent down to the device.
4) For a windowing EDD application, if a METHOD is running while another configuration window is open, any changes to VARIABLES in the METHOD won't be reflected in the open configuration window until the value is committed.
5) For a windowing EDD application, if a METHOD is running while another configuration window is open, when the METHOD causes a new value of a parameter to be read from the device, the configuration window will reflect the updated value. This update may occur after the method execution is complete.
6) While in a METHOD, references to VARIABLES of class DYNAMIC shall not cause new values to be read continuously from the device unless the DISPLAY builtin or format option "D" is used (see "Format Options" table in IEC 61804-5).
7) Whenever a METHOD utilizes any UI Builtin functions (see IEC 61804-5) that can accept input from the user, the ability to abort the METHOD shall also be provided to the user. The only exception should be when an abort METHOD is running, as abort methods cannot be aborted.
8) When a METHOD terminates, either normally or through an abort, the handling of data in the method cache can be controlled by calling the method cache controlling Builtins (see 8.2.5.5).
9) When an action method terminates, all VARIABLES modified via the fsetval, isetval, or ssetval Builtin functions shall be automatically retained in the primary cache. Caching of all other data-item values shall adhere to Rule 8.
10) While in a METHOD, any UI builtin called that results in an edit or refresh action aborting the UI builtin shall terminate and return an error to the calling METHOD. If the UI builtin can not return an error code, then the UI function will show any defined static text but not the variable referenced that caused the edit or refresh action to abort.

6.3 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 an option list in a method. Instead of displaying text strings, the Builtin MenuDisplay will display the specified menu and append the user selected options 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.

Communication profiles: FF, HART, ISA100_Wireless

The number of entries in the selection list is limited to three.

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 shall 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.

Communication profiles: FF, HART, ISA100

METHODs and EDIT_DISPLAYs are not permitted MENU items (including referenced submenus, if any appears on a MENU called from MenuDisplay).

Figure 67 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 67 – Example of a wizard

6.4 Division by zero and undetermined floating values

6.4.1 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.

6.4.2 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 29 shows examples of results of floating-point calculations, whereas ‘n’ is a positive numeric value.

Table 29 – 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 with indications such as “NaN” or “Not a Number”. Infinity shall be displayed with indications such as “Inf” or “Infinity” or infinity symbol. The negative sign of infinity shall also be displayed. The display of positive sign of infinity is optional.

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 shall be contained within the modular device, i.e. a module cannot 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 30.

Table 30 – 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 68.

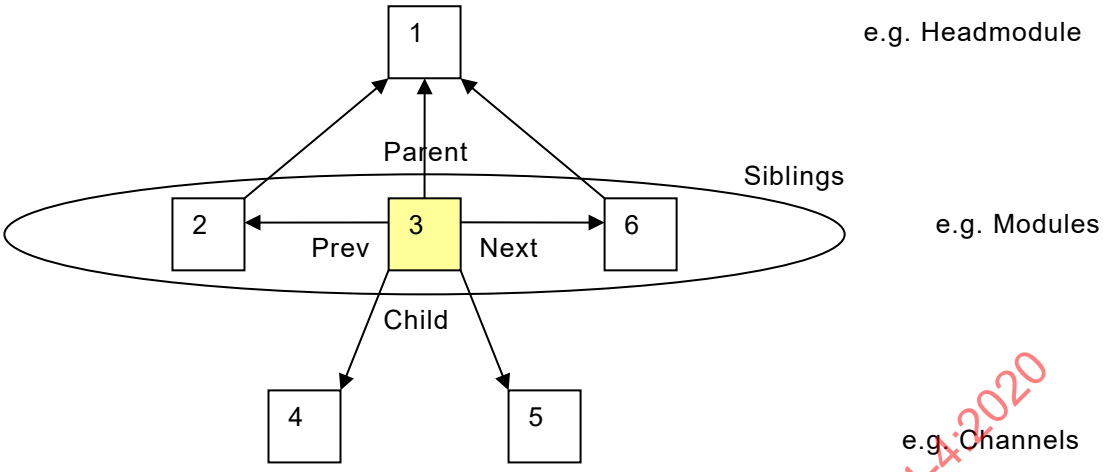


Figure 68 – The different relations of a module

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 7.9.

7.6 Simple modular device example

7.6.1 General

In the examples in Figure 69, a simple modular device is described that allows to plug two types of modules into two slots.

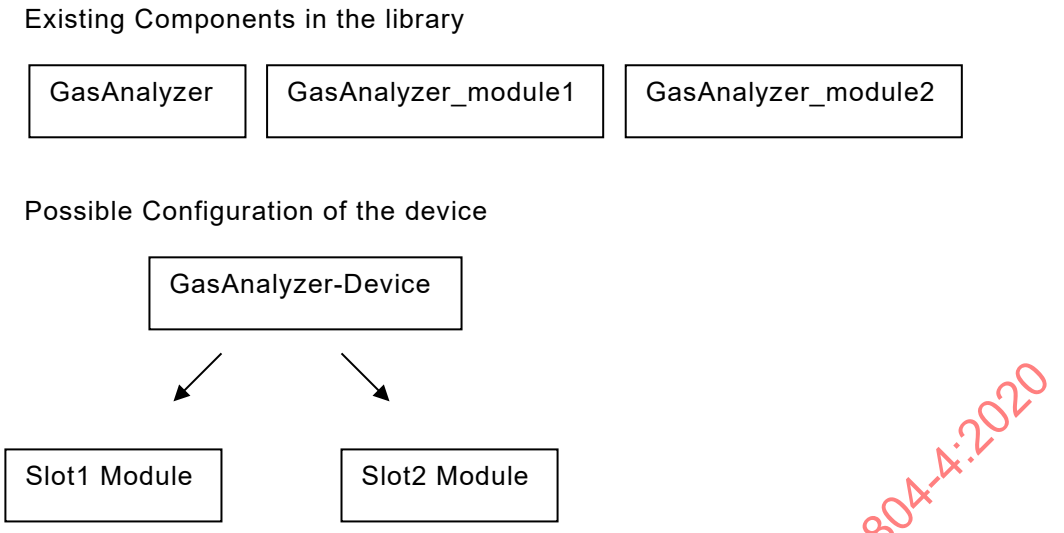


Figure 69 – Components and possible configuration of the modular devices

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 70 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 70 – Separate EDD file example with direct EDD referencing

Figure 71 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 71 – EDD example for module1

Figure 72 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 72 – EDD example for module2

7.6.3 Separate EDD file example with classification EDD referencing and interfaces

Figure 73 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 73 – EDD example for modular device

Figure 74 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 74 – EDD example for module1

Figure 75 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 75 – 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 76).

```
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 76 – 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.

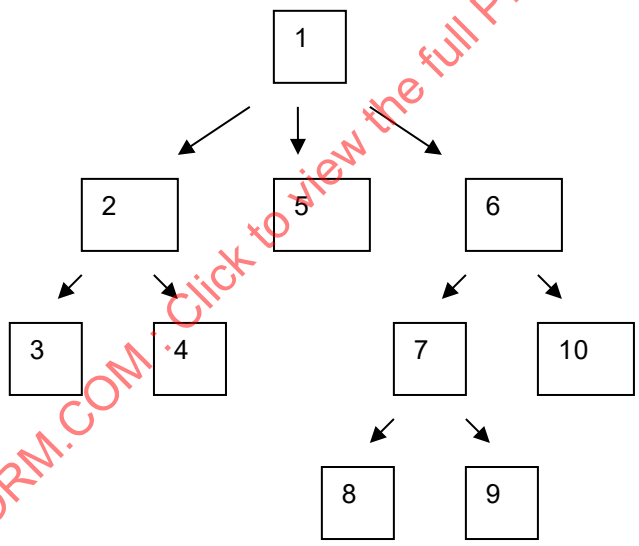
7.7 Upload and download for modular devices

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 device.

7.8 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 31.

Table 31 – 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.	

7.9 Reading modular device topology

7.9.1 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.

7.9.2 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.

7.10 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 configuration check 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 Session management

8.1 Overview

Session management describes the EDD sessions handled by an EDD application. This section also describes caching of data, dependencies between the caches and impact on user interface.

Table 32 provides the definitions for terminology related to session management.

Table 32 – Terminology for session management

Term	Definition
Session	The period during which the user is working on an instance of the EDD by means of the EDD application.
Edit session	The period of time between the beginning of modifying data in an edit cache and committing those changes to the parent cache.
Online session	A session requiring a connection to the device.
Offline session	A session based on offline dataset.

8.2 Data management

8.2.1 Overview

Table 33 provides the definitions for terminology related to data management.

Table 33 – Terminology used in data management

Term	Definition
Dataset	Set of data containing the unscaled values of all data items of a specific device type instance.
Configuration store	Persistent storage where the EDD application may store the dataset of the offline session.
Cache	Collection of changed data item values to separate the data modifications of a session.
Offline cache	Dataset containing the values of data items used for offline session.
Online cache	Collection of values of data items from the device.
Edit cache	Subordinated cache based on the invoking parent cache (cache that the edit cache is subordinate to) for windows and dialogs.
Method cache	Subordinated cache based on the invoking parent cache (cache that the method cache is subordinate to) for executing methods. This can be considered as a specialized edit cache.
Parent cache	Cache that the edit cache or method cache is subordinate to.

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 cache values in the current cache.

Communication profiles: FF, ISA100

For FF and ISA100, caches shall support multiple block instances.

8.2.2 Caching for online session

EDD application shall use online cache during online session. Edit caches and method caches may be created during the online session.

Values of data, except VARIABLEs with HANDLING WRITE (“write-only” variables), or with CLASS LOCAL, LOCAL_A and LOCAL_B in the online cache shall be initialized with data read from the device.

Communication profiles: HART, ISA100, FF

Values of VARIABLEs of CLASS LOCAL, LOCAL_A and LOCAL_B in the online cache shall be initialized with initial values as defined in 5.3.2.

Communication profiles: CS, GPE, PB, PN

Values of VARIABLEs of CLASS LOCAL, LOCAL_A and LOCAL_B in the online cache shall be initialized from the offline cache. If offline cache does not exist, then the values shall be initialized with initial values as defined in 5.3.2.

Content of FILE objects shall be initialized from a means of persistent storage.

Values of data, except VARIABLEs of type LOCAL, LOCAL_A and LOCAL_B shall be written to the device after a user chooses to commit them.

After committing to the device, the committed variables and the affected VARIABLEs of UNIT and REFRESH relations shall be reread if needed. As a minimum, the dependent items shall be reread prior to using or displaying any of the dependent variables in the relation, but after any modified values have been written to the device.

Content of FILE objects shall be committed to a means of persistent storage.

Figure 81 shows the data caching for an online session.

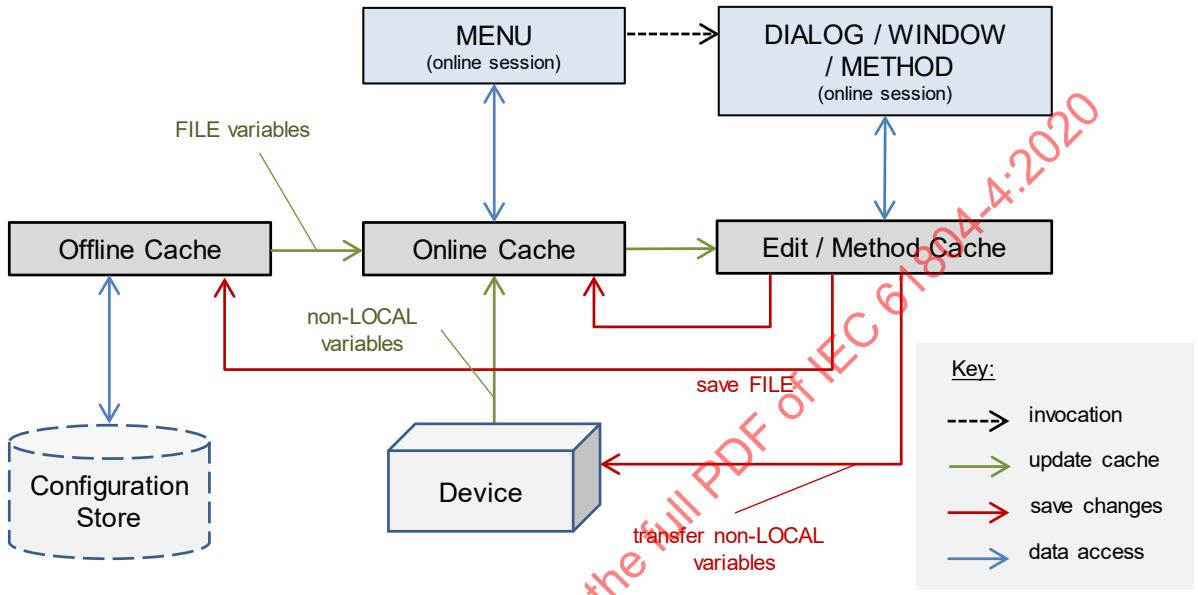


Figure 81 – Data caching for an online session

8.2.3 Caching for offline session

EDD application shall use offline cache during offline session. Edit caches and method caches may be created during the offline session.

Values of data in the offline cache shall be initialized from the configuration store. If the configuration store does not exist, then the values shall be initialized with initial values as defined in 5.3.2.

Values of all data shall be saved to the offline cache after a user chooses to commit them.

Figure 82 shows the data caching for an offline session.

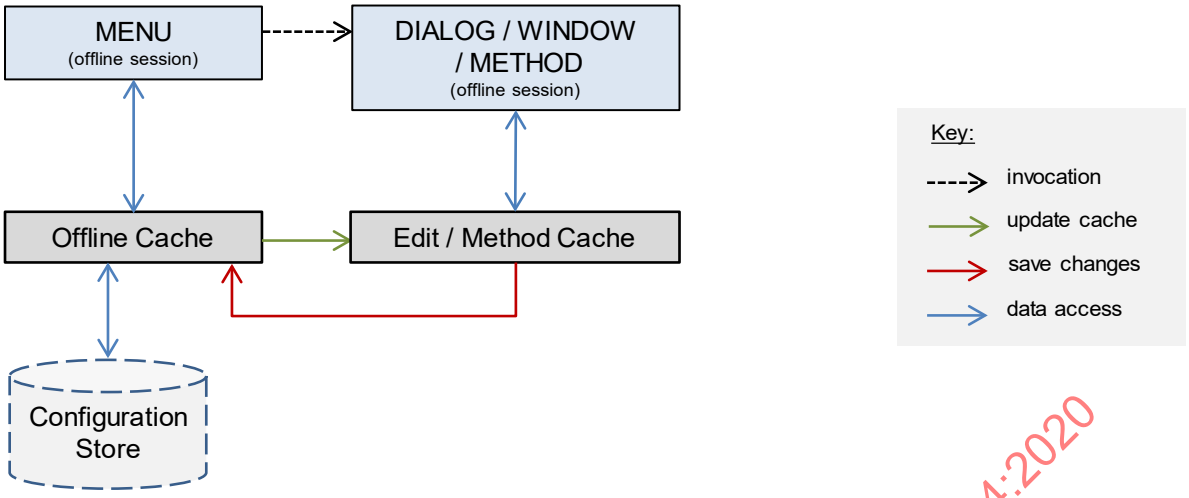


Figure 82 – Data caching for an offline session

8.2.4 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 which is initialized with required data values from the parent cache.

Values of required data shall be committed to the parent cache upon committing.

EDD applications may use either shared edit caches or separate edit caches for sub windows and dialogs.

While using shared edit caches, 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.

While using separate edit caches, the changes are committed to the parent cache, if the sub dialog or window is confirmed, otherwise the changes are discarded.

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

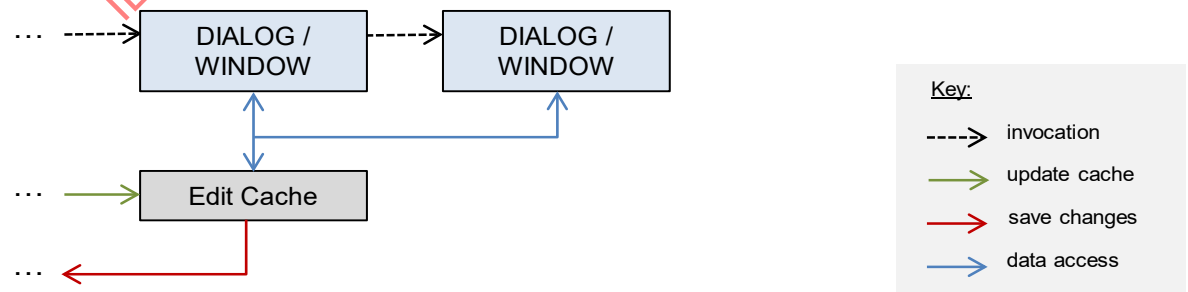


Figure 83 – Sub dialogs or windows using a shared edit cache

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

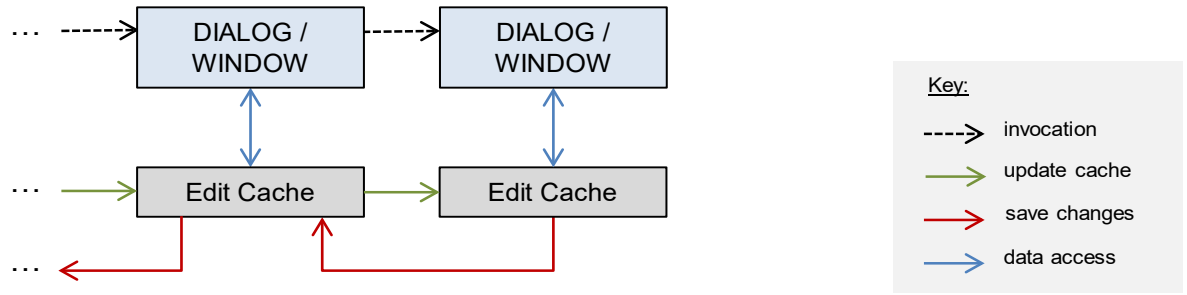


Figure 84 – Sub dialogs or windows using separate edit caches

8.2.5 Caching for METHODs

8.2.5.1 General

When the user invokes a METHOD, the EDD application shall use a method cache with the required values based on the parent cache. During an online session, values of device VARIABLEs are read on demand from the device, i.e. when requested while the method is executing. METHOD local variables are not stored in the method cache. If a METHOD aborts, all changes in the method cache which are not already saved using the save_values Builtin shall be discarded. The saving of changes within a METHOD is described in Table 34.

8.2.5.2 Caching for nested METHODs

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 which are not already saved using the save_values Builtin shall be discarded. The saving of changes within a METHOD is described in Table 34. Figure 85 shows the data caching for nested METHODs.

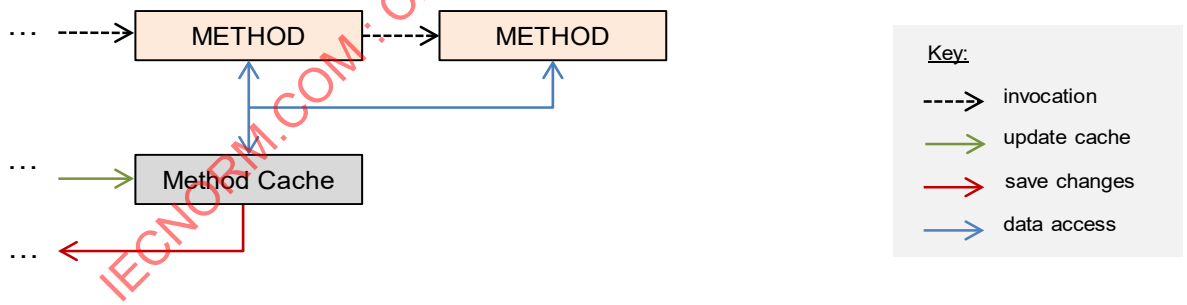


Figure 85 – Data caching for nested METHODs

8.2.5.3 Caching for METHOD invoked from dialog or window

When a METHOD is invoked from a dialog or window, the parent cache of the method cache shall be the edit cache of the dialog or window from which the METHOD was invoked.

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

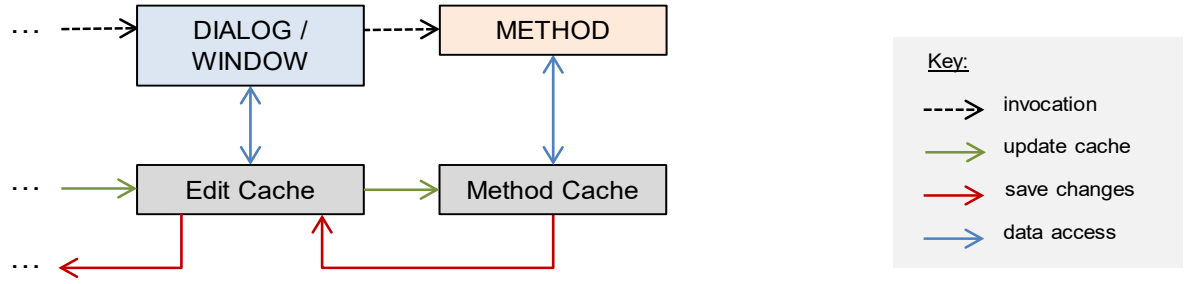


Figure 86 – Data caching for a METHOD invoked within a dialog or window

8.2.5.4 Caching for METHOD invoking a dialog

When a dialog is invoked from a METHOD by calling the Builtin MenuDisplay, the EDD application may either use the method cache of the calling method or may use a separate 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.

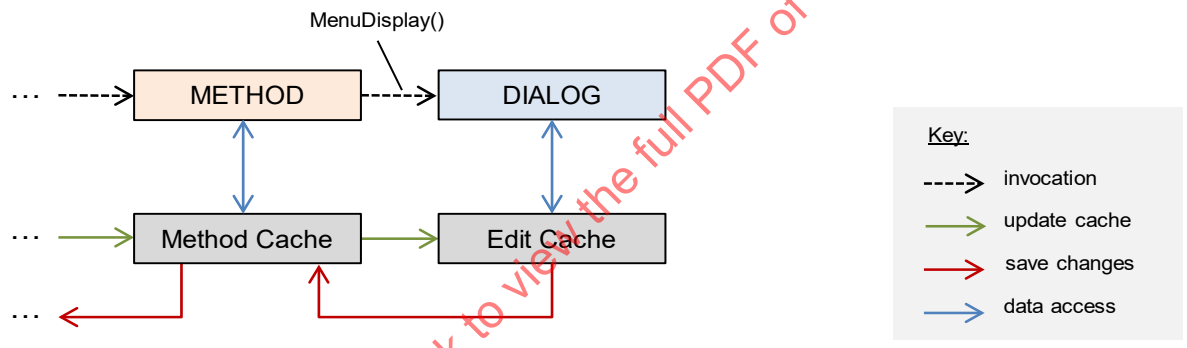


Figure 87 – Data caching for a METHOD invoking a dialog using an edit cache

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

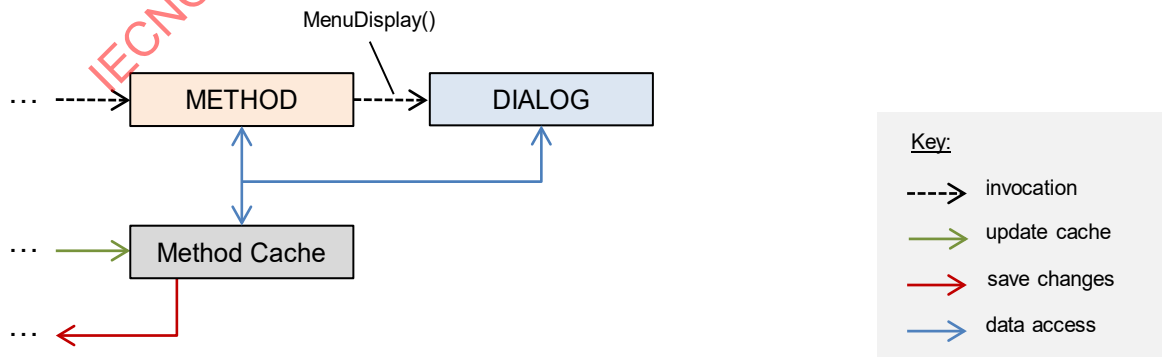


Figure 88 – Data caching for a METHOD invoking a dialog

Communication profiles: HART, ISA100, FF

HART, FOUNDATION fieldbus and ISA100_Wireless do not support invoking a METHOD from a dialog, which is invoked from a METHOD by calling the Builtin MenuDisplay.

8.2.5.5 Committing data from METHOD

In METHODs invoked by the user, the Builtins save_values, save_on_exit, send_on_exit, send_all_values or send_value control saving the changes of the method cache. Table 34 shows the usage of the Builtins (see IEC 61804-5). A METHOD can also transfer values from and to the device by calling communication Builtins.

Table 34 – Builtins for method cache controlling

Builtin	HART	FOUNDATION fieldbus, ISA100_Wireless	Communication , GPE, PROFIBUS, PROFINET	Cache handling
save_values	Required	—	Optional	All changes shall be saved to calling cache immediately.
save_on_exit	—	Required	Default behavior	All changes shall be saved to calling cache when the method finishes correctly.
discard_on_exit	Default behavior	Optional	Optional	Discards all changed values at the end of the method. Calling cache is not modified.
send_on_exit	—	Optional	Optional	All changes shall be saved to calling cache and transferred to the device when the method finishes correctly.
send_all_values	—	Optional	—	All changes shall be immediately saved to calling cache and transferred to the device.
send_value	—	Optional	—	The value shall be immediately saved to calling cache 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.

VARIABLE values in the method cache which are to be discarded shall only be discarded after the execution of the last abort method (if any) is completed and the METHOD terminates.

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.

Any conditionals shall be evaluated using the cache values in the current cache.

Communication profiles: FF, ISA100

For FF and ISA100, caches shall support multiple block instances.

8.3 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. ENUMERATED and BIT_ENUMERATED variables with no related enumerator shall display the current value numerically.

b) VARIABLEs shall be marked, if the value is changed by the user or indirectly by METHODs.
The EDD application may show additional indications. After transfer to the device or save to the offline data base, or a variable is reread from the device, a variable is no longer marked.

c) 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 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.

d) The EDD application may allow that the user enters invalid numeric values that are outside the MIN/MAX_VALUE ranges (out-of-range) but are inside the range of the data type to support complex dependences. If this occurs, the EDD application shall provide an indication within close proximity, e.g. next to variable on the same line of the menu, that the variable value is out-of-range. This indication shall be consistently used throughout the EDD application for all out-of-range values. This indication shall appear as long as the value is out-of-range. Out-of-range values may be saved to and restored from a configuration file. However, if the value is still out-of-range when restored, the indicator shall be shown again. If out-of-range VARIABLES are attempted to be committed to the device, a message indicating non-commitment shall be shown that includes the out-of-range VARIABLEs "label". While editing ENUMERATED and BIT_ENUMERATED VARIABLEs, the user shall not be able to enter invalid values.

e) When a WINDOW instance is invoked from a data cache that already has the same WINDOW open, then the existing WINDOW instance shall be activated. When a WINDOW instance is invoked from a data cache that does not have the same WINDOW open, then a new WINDOW instance shall be activated. An example of referencing a WINDOW from a different data cache could be when a WINDOW is opened from an online context and again from an offline context.

f) All sub windows of a DIALOG shall be handled like a DIALOG.

g) A MENU of STYLE DIALOG is modal; that means no interactions outside of the DIALOG shall be possible.

h) If a cause variable of a UNIT or REFRESH relation is changed, the EDD application shall mark the dependent variables. In an online session, the modified values shall only be written to the device when commit (or similar) button is pressed.

i) If a dependent variable of a UNIT or REFRESH relation is changed, the EDD application may mark the dominant variable.

j) Some EDD applications may support unit conversion for recalculating dependent variable values automatically whenever a dominant unit code variable is modified. EDD applications that support unit conversion should only apply defined and compatible unit conversion formulas (e.g. IEC, NIST, GOST, and other recognized standards).

8.4 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 parameters 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. Variables of class dynamic are not stored as part of a device's configuration.

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.

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

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

In order to allow the user to navigate to MENUs or METHODs with ACCESS OFFLINE referenced from an online configuration menu, the communication should be attempted as late as possible, that means only when data is required, e.g. for evaluating conditionals.

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. Annex D shows the upload and download caching model.

Furthermore, two types of upload and download procedures shall be supported: Non-interactive and interactive. In non-interactive upload and download, the host transfers data from and to devices without requiring user guidance to control the data transfer. An interactive upload and download may include user guidance to control the transfer of the data from and to devices.

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.

METHODs invoked while performing non-interactive uploading or downloading shall not require any user interactions.

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

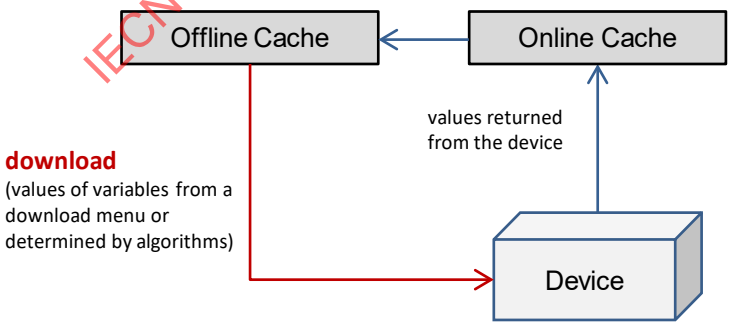


Figure 89 – Data flow for download to the device

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

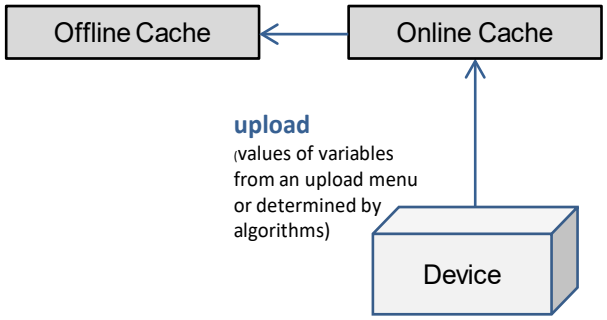


Figure 90 – Data flow for upload from the device

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. The METHODs invoked on these MENUs shall not contain user interaction for non-interactive upload and download. The METHODs invoked on these MENUs may contain user interaction for interactive upload and download.

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 shall 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 shall not be considered to 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 35 in order to determine which data items should be read from the device. The EDD application shall support the identifiers in Table 35.

Table 35 – List of defined upload menu identifiers

Menu identifiers
upload_from_device_root_menu
download_variables

They shall be used according to the following criteria:

- If the upload_from_device_root_menu and the download_variables menu exists in the EDD, the upload_from_device_root_menu shall be used for non-interactive upload and the download_variables menu shall be used for interactive upload.

- If only the upload_from_device_root_menu exists in the EDD, the EDD application shall use it for both interactive and non-interactive upload.
- If only the download_variables menu exists in the EDD, the EDD application shall support interactive upload. The EDDL application shall use the algorithm defined in 9.6.3.4 for non-interactive upload.
- If the EDD does not have the upload_from_device_root menu and does not have the download_variables root menu, the EDD application shall use the algorithm defined in 9.6.3.4.

9.6.3.2 General rules

The following are the general rules for the upload procedure.

- CLASS LOCAL variables used in conditional expressions and methods referenced by the upload menu shall be initialized with values from the Offline Cache.
- 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 interpret 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 that appears on the upload menu should not be read. A VARIABLE with VALIDITY FALSE that is used in a conditional expression shall be attempted to be read in an attempt to evaluate the expression.
- The PRE_READ_ACTIONS of the VARIABLES shall be executed immediately before and the POST_READ_ACTIONS after reading the VARIABLE from the device. User interaction is not allowed when VARIABLES containing these actions are placed on the upload_from_device_root_menu.
- 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.
- UNIT and REFRESH relations are not executed during upload.
- If user interaction is detected on a method associated with the upload_from_device_root_menu for a non-interactive upload, then the EDD application shall record the user interaction request and shall discontinue upload to that device by default operation.

9.6.3.3 Upload with upload menu

If an upload menu is defined, the upload shall proceed in the following steps for both non-interactive and interactive uploads. Figure D.1 illustrates the caching and data flows for methods and actions during upload.

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.
4) Only MENUs and data items (VARIABLEs, RECORDs, etc.) shall be permitted on upload menus.
5) METHODs shall not be permitted on upload menus. The EDD writer should use the read actions of the menu for any business logic.
6) EDD applications should ignore any invalid menu items.

If an EDD defines the MENU upload_from_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 affected-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.

9.6.4 Download procedure

9.6.4.1 Overview

An EDD may contain a download menu by using one of the identifiers in Table 36 in order to determine which data items should be written to the device.

The EDD application shall support the identifiers in Table 36.

Table 36 – List of defined download menu identifiers

Menu identifiers
download_to_device_root_menu
upload_variables

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

The menu identifiers shall be used according to the following criteria:

- If the download_to_device_root_menu and the upload_variables menu exists in the EDD, the download_to_device_root_menu shall be used for non-interactive download and the upload_variables menu shall be used for interactive download.

- If only the download_to_device_root_menu exists in the EDD, the EDD application shall use it for both interactive and non-interactive download.
- If only the upload_variables menu exists in the EDD, the EDD application shall support interactive download. The EDDL application shall not use upload_variables for non-interactive download. A non-interactive download shall not be initiated and error should be logged.
- If the EDD does not have the download_to_device_root menu and does not have the upload_variables root menu, the EDD application shall use the algorithm defined in 9.6.4.4.

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. User interaction is not allowed when VARIABLES containing these actions are placed on the download_to_device_root_menu.
- 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.
- If user interaction is detected on a method associated with the download_to_device_root_menu for a non-interactive download, then the EDD application shall record the user interaction request and shall discontinue download to that device by default operation.

9.6.4.3 Download with download menu

If a download menu is defined, the download shall proceed in the following steps for both non-interactive and interactive downloads. Figure D.2 illustrates the caching and data flows for Methods and Actions during download.

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 according to 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) If a variable that is not writeable (e.g. it is not present in any write command) is encountered on the download menu, that variable shall be skipped and the download continues with the next variable on the menu.

5) When using the upload_variables menu for HART, all remaining writable data items shall also be downloaded to the device.

6) Only MENUs and data items (VARIABLEs, RECORDs etc.) shall be permitted on download menus.

7) METHODs shall not be permitted on download menus. The EDD writer should use the write actions of the menu for any business logic.

8) EDD applications should ignore any invalid menu items.

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", except VARIABLEs of CLASS LOCAL or DYNAMIC. The cause-parameters of UNIT and REFRESH relations should be written before the affected-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 affected-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 and ISA100_Wireless 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.2) COLLECTION as specified for each BLOCK_A COLLECTION_ITEMS attribute shall be not part of a download.

10 EDDL communication description

10.1 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 when the read fails.

10.2 Parsing data received from the device

Communication profiles: FF, ISA100

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.

Communication profiles: 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 when the device does not respond or the device responds with an error response.

Communication profiles: PB, PN, CS, GPE

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 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

10.4 Foundation Fieldbus

FOUNDATION fieldbus devices are block object oriented devices according to IEC 61804-2. Each block is composed of a characteristic 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 91 illustrates 2 instances of BLOCK b1 and 1 instance of BLOCK b2 in an object dictionary. The EDDL fragment shown in Figure 92 can be used to describe this example.

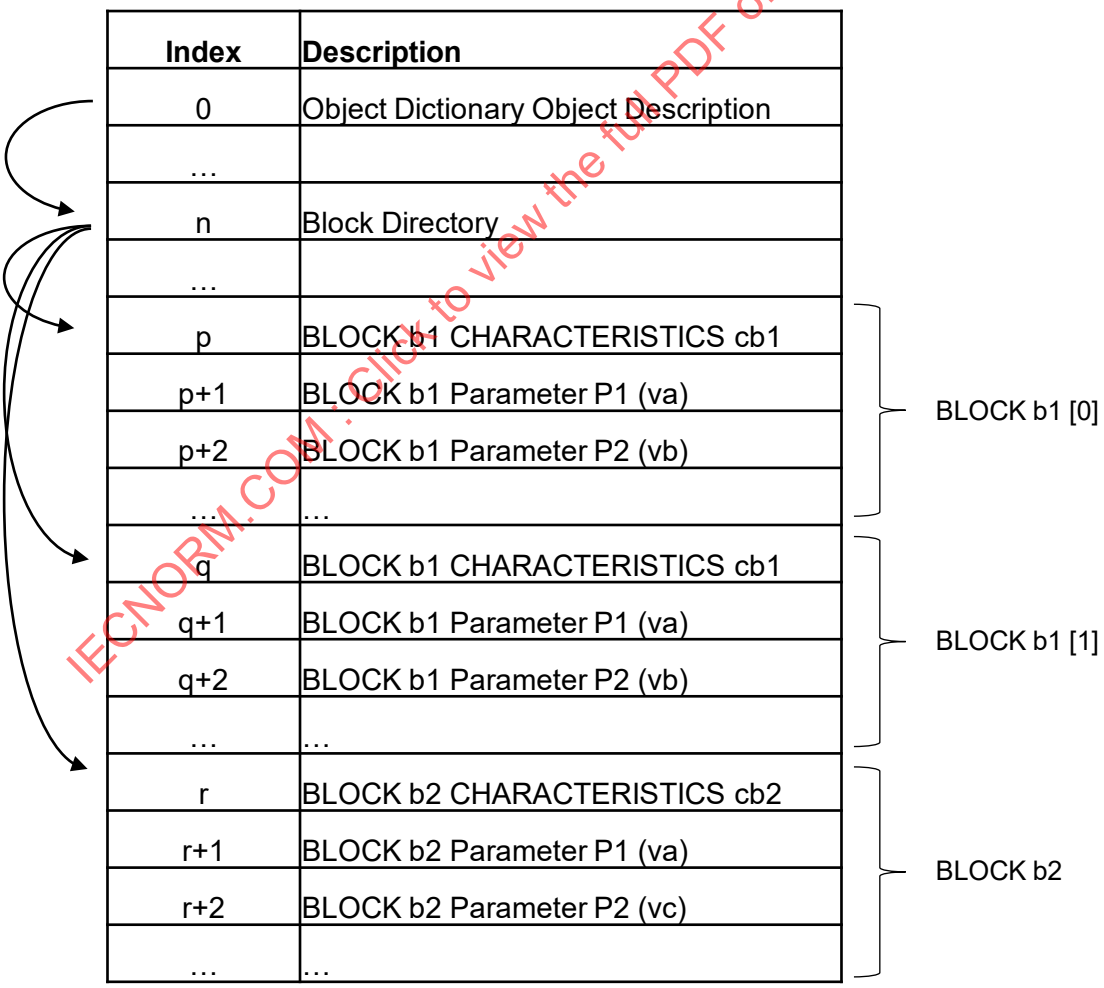


Figure 91 – Example device with 2 unique BLOCK_A definitions

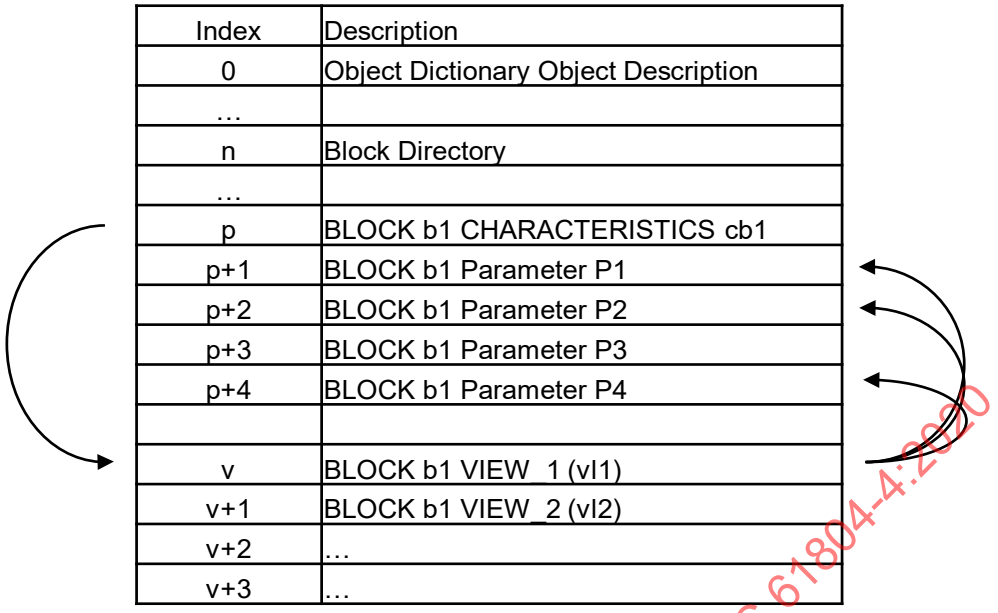
```
BLOCK b1
{
    CHARACTERISTICS cb1;
    PARAMETERS
    {
         P1, va;
         P2, vb;
         /* ... */
     }
}

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 92 – 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 93 illustrates an example of VARIABLE_LIST vl1 for BLOCK b1. The EDDL fragment shown in Figure 94 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 93 – 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 94 – Example EDD for a BLOCK_A with PARAMETER_LISTS

10.5 ISA100_Wireless communication model

ISA100_Wireless devices are block object oriented devices according to IEC 61804-2. Each block is composed of one or more block parameters. In EDDL, each block type is defined using BLOCK_A construct. Block parameters can be VARIABLEs, RECORDs and VALUE_ARRAYs.

ISA100_Wireless device supports exactly one User Application Process Management Object (UAPMO) block object, according to IEC 62734:2014. One or more block objects are referenced from a UAPMO block object. In a ISA100_Wireless device, block object parameters are termed as attributes and have a unique attribute number or id within a block object. The attributes in UAPMO are accessed by attribute number using read and write services. UAPMO should be referenced for the block object id's. UAPMO has an attribute called "Array of ObjectIDandType", (attribute no. 9) to indicate list of block objects available within the device.

All data items accessible by an EDD application are accessed in the context of a block object. The EDD does not provide any information about the attribute ids of the item's in the block objects. Instead, block parameters provide reference to the EDD item using a unique symbol id.

When an EDD is created, unique symbol ids are generated for each EDD item. These symbol id's in the EDD are associated to the corresponding block object id's and the attribute id's in the device.

By accessing the block to find the associated symbol id, the EDD application associates an EDD BLOCK_A declaration with the block object in the device. All BLOCK_A PARAMETERS are listed consecutively in the EDD but the corresponding attribute id's, in the device, may not be sequential (and should be referenced from the CFF file, which is defined in ISA100.11a. By accessing the attribute id's, the EDD application can access each BLOCK_A PARAMETERS.

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

Figure 95 illustrates an ISA100_Wireless device having 2 Analog Input blocks. These block ids are known by reading the UAPMO attribute 9 information using read service. The EDDL fragment in Figure 96 can be used to describe this example.

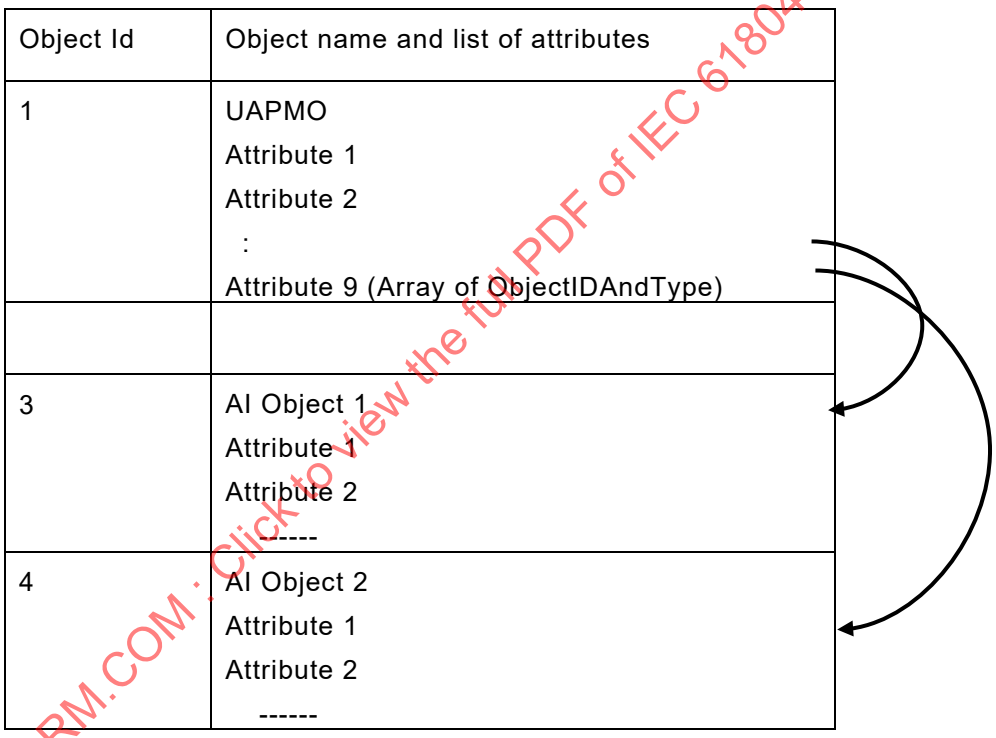


Figure 95 – Example ISA100_Wireless device objects representation

```
BLOCK b1
{
    PARAMETERS
    {
         P1, va;
         P2, vb;
         /* ... */
     }
}

BLOCK b2
{
    PARAMETERS
    {
         P1, va;
         P2, vc;
         /* ... */
     }
}

VARIABLE va { /* ... */ }
VARIABLE vb { /* ... */ }
VARIABLE vc { /* ... */ }
```

Figure 96 – Example EDD for a ISA100_Wireless device with 2 unique BLOCK_A definitions

Figure 97 illustrates an example of PARAMETER LIST for BLOCK b1. The EDDL fragment in Figure 98 can be used to describe this example.

Parameter type	Parameter index	Description
Standard parameters	1	BLOCK b1 Parameter 1
	2	BLOCK b1 Parameter 2
	3	BLOCK b1 Parameter 3
	:	:
Profile parameters	26	BLOCK b1 Parameter 4
	27	BLOCK b1 Parameter 5
	28	BLOCK b1 Parameter 6
	:	:
Vendor parameters	64	BLOCK b1 Parameter 7
	65	BLOCK b1 Parameter 8
	66	BLOCK b1 Parameter 9
	:	:

Figure 97 – BLOCK_A example with PARAMETER_LISTS

The parameters in the EDD file are defined sequentially in the order, first standard parameters followed by profile parameters followed by vendor parameters. The start and end of the parameter indexes of each parameter type in the BLOCK b1 are defined in the CFF file for EDDL application to extract the parameter indexes.

```
BLOCK b1
{
    PARAMETERS
    {
         P1, va;
         P2, vb;
         P3, vc;
         P4, vd;
         P5, ve;
         P6, vf;
         P7, vg;
         P8, vh;
         P9, vi;
         P10,vj;
         P11,vk;
         P12,vl;
    }
};

VARIABLE va { /* ... */ }
VARIABLE vb { /* ... */ }
VARIABLE vc { /* ... */ }
VARIABLE vd { /* ... */ }
VARIABLE ve { /* ... */ }
VARIABLE vf { /* ... */ }
VARIABLE vg { /* ... */ }
VARIABLE vg { /* ... */ }
VARIABLE vi { /* ... */ }
VARIABLE vj { /* ... */ }
VARIABLE vk { /* ... */ }
VARIABLE vl { /* ... */ }
```

Figure 98 – Example EDD for a BLOCK_A with PARAMETER_LISTS

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 background_period VARIABLE.

The 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. 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 lists predefined identifiers that host applications can use to access standardized arrays.

Table B.1 – ARRAY predefined identifiers

Item type	Identifier	Profile	Description
ARRAY OF VARIABLE	factory_protection_array	HART	This lists VARIABLEs that are not copied from one device instance into new instances.
ARRAY	loop_warning_variables	HART	This ARRAY contains a list of variables, that when changed and the Send/Commit key is pressed cause a warning message to appear. This variable list shall include all variables that could disrupt, or change the output loop current when they are sent to and stored in the field device.

Table B.2 lists predefined identifiers that host applications can use to access standardized collections.

Table B.2 – COLLECTION predefined identifiers

Item type	Identifier	Profile	Description
COLLECTION	no_download*	FF, ISA100	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)
COLLECTION	upload_wanted*	FF, ISA100	Read-only COLLECTION of BLOCK_A PARAMETERS that should be included in an offline dataset
Key: *: postfix for identifiers			

Table B.3 lists predefined identifiers that host applications can use to access standardized commands.

Table B.3 – COMMAND predefined identifiers

Item type	Identifier	Profile	Description
COMMAND	read_additional_device_status	HART	This COMMAND (48) should be dispatched when "More Status Available" is set in the "Device Status" byte. The meaning of the command response is defined in the DD currently being evaluated. Support for Command 48 was optional prior to the introduction of HART 7.

Table B.4 lists predefined identifiers that host applications can use to access standardized images.

Table B.4 – IMAGE predefined identifiers

Item type	Identifier	Profile	Description
IMAGE	device_icon	HART	An IMAGE for a device

Table B.5 lists predefined identifiers that host applications can use to access standardized menus.

Table B.5 – MENU predefined identifiers

Item type	Identifier	Profile	Description
MENU	device_root_menu	HART, FF, PB, PN, ISA100, CS, GPE	See 4.3.
MENU	device_root_menu*	FF, ISA100	BLOCK_A based MENU optimized for PC based applications.
MENU	diagnostic_root_menu	HART, FF, PB, PN, ISA100, CS, GPE	See 4.3.
MENU	diagnostic_root_menu*	FF, ISA100	BLOCK_A based MENU optimized for PC based applications.
MENU	download_to_device_root_menu	HART, FF, PB, PN, ISA100, CS, GPE	See 9.6.4.
MENU	download_variables	PB, PN	See 9.6.3.1.
MENU	hh_device_root_menu	FF, ISA100	Device level MENU optimized for handheld applications.
MENU	hh_diagnostic_root_menu	FF, ISA100	Device level MENU optimized for handheld applications.
MENU	hh_process_variables_root_menu	FF, ISA100	Device level MENU optimized for handheld applications.
MENU	hot_key	HART	This is a short cut MENU that allows a user access to frequently used DD Items.
MENU	maintenance_root_menu	HART, FF, PB, PN, ISA100, CS, GPE	See 4.3.
MENU	Menu_Top*	FF, ISA100	BLOCK_A MENU optimized for handheld applications.
MENU	Menu_Main_Maintenance[a]	PB	MENU added to the menu bar of the EDD application.
MENU	Menu_Main_Specialist[a]	PB	MENU added to the menu bar of the EDD application.
MENU	offline_root_menu	HART, FF, PB, PN, ISA100, CS, GPE	See 4.3.
MENU	OnlineWindow_display[a]	PB	MENU containing process variables.
MENU	process_variables_root_menu*	FF, ISA100	BLOCK_A based MENU optimized for PC based applications.
MENU	process_variables_root_menu[a]	HART, FF, PB, PN,	See 4.3.

Item type	Identifier	Profile	Description
		ISA100, CS, GPE	
MENU	root_menu	HART	See 4.2.
MENU	Table_Main_Maintenance[a]	PB	A starting point for the explorer view of a device.
MENU	Table_Main_Specialist[a]	PB	A starting point for the explorer view of a device.
MENU	upload_from_device_root_menu	HART, FF, PB, PN, ISA100, CS, GPE	See 9.6.3.
MENU	upload_variables	HART, PB, PN	See 9.6.4.
MENU	view_menu	HART	This MENU shall only be supported in the SDC-625. This allows access to SDC-625 "View" menu (on SDC-625 menu bar). It shall only be used to facilitate testing and development of field device DDs.
Key: *: postfix for identifiers			
[a] These identifiers should not be used to write new EDDs.			

Table B.6 lists predefined identifiers that host applications can use to access standardized methods.

Table B.6 – METHOD predefined identifiers

Item type	Identifier	Profile	Description
METHOD	device_simulation_method	HART	See Annex A.
METHOD	GetHealthStatus()	HART, FF, PB, PN, ISA100, CS, GPE	Used by FDI servers to retrieve the condensed device health (see FDI Specifications).

Table B.7 lists predefined identifiers that host applications can use to access standardized variables.

Table B.7 – VARIABLE predefined identifiers

Item type	Identifier	Profile	Description
VARIABLE	comm_status	HART	When bits are set in this VARIABLE there has been a communication error.
VARIABLE	background_period	HART	See Annex A.
VARIABLE	device_status	HART	A bit set in this VARIABLE indicates a potential problem in the device.
VARIABLE	extended_device_status	HART	A bit set in this VARIABLE indicates a potential problem in the device.
VARIABLE	hart_functions	HART	Indicates EDD application special features.
VARIABLE	response_code	HART	This VARIABLE contains the response code for the command response received.
VARIABLE	std_ResponseCode	PB, PN	Standard communication response for PROFIBUS and PROFINET.

Annex C
(informative)

Description of EDDL profiles

C.1 Communication Server (CS)

An FDI communication server (IEC 62769-7) provides a standardized interface to a protocol driver for an FDI host. It provides access to single filed devices or field device networks. An FDI communication package (IEC 62769-4) for an FDI communication server provides the capability to configure such an FDI communication server (e.g. to configure the baud rate, preambles, retries). The EDD profile for an FDI communication server (CS) defines the allowed language concepts, rules, etc. for the EDD contained in such an FDI communication package for an FDI communication server.

C.2 Foundation Fieldbus (FF)

The EDD profile for Foundation Fieldbus defines the allowed language concepts, rules, etc. for an EDD designed to work with a Foundation Fieldbus instrument.

C.3 Generic Protocol Extension (GPE)

FDI defines a way to generically add protocols to existing FDI hosts (IEC 62769-100). It defines an FDI profile that has a well-defined interface for an FDI communication server and uses the EDD profile Generic Protocol Extension (GPE) for FDI device packages. The GPE defines the allowed EDD language concepts, rules, etc. for all EDDs following this approach. This allows the FDI host to deal the same way for all generically added protocols. The GPE uses the HEADER attribute for addressing information in the EDD. The format of the HEADER is protocol-specific and defined in a protocol-specific definition (PSD, see IEC 62769-100). As the translation of the address information to the concrete protocol is done in the protocol driver (the FDI communication server) and all protocol drivers have the same interface the FDI host does not need to know the protocol in advance and can just integrate the protocol driver in order to support a new protocol. The FDI communication server and the FDI device packages for the protocol need to follow the PSD of the protocol. An example of a PSD is Modbus-RTU (IEC 62769-115-2).

C.4 HART

The EDD profile for HART defines the allowed language concepts, rules, etc. for an EDD designed to work with a HART instrument.

C.5 ISA100

This profile for ISA100 defines the allowed language concepts, rules, etc. for an EDD designed to work with an ISA100 wireless instrument.

C.6 PROFIBUS (PB)

This profile for PROFIBUS defines the allowed language concepts, rules, etc. for an EDD designed to work with a PROFIBUS instrument.

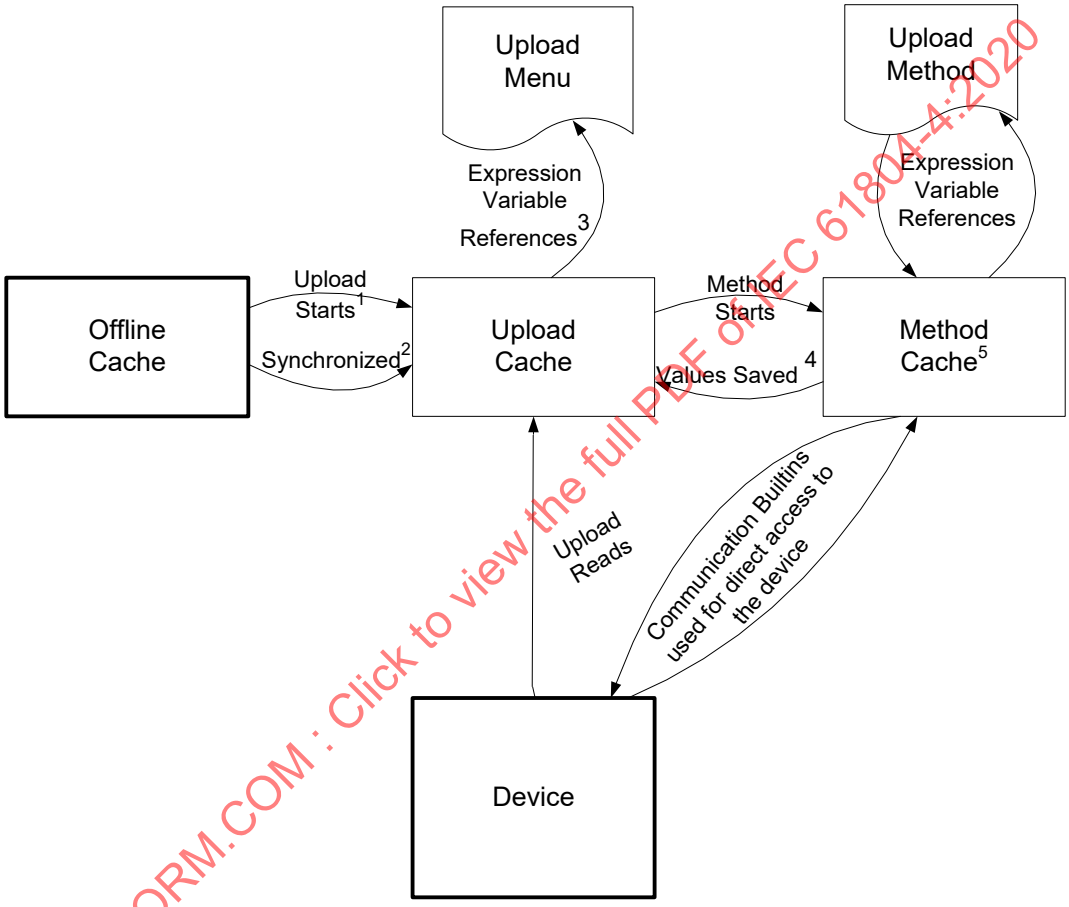
C.7 PROFINET (PN)

This profile for PROFINET defines the allowed language concepts, rules, etc. for an EDD designed to work with a PROFINET instrument.

Annex D
(normative)

Upload/download caching model

Both the Offline and Online datasets are active during an Upload or Download making the initialization of caches less obvious. The diagrams in Figure D.1 and Figure D.2 illustrate the proper initialization and data flow through the various caches during and Upload or Download.



Key

1 Upload cache LOCAL variables are initialized from the offline cache. All others are initialized from the online cache.

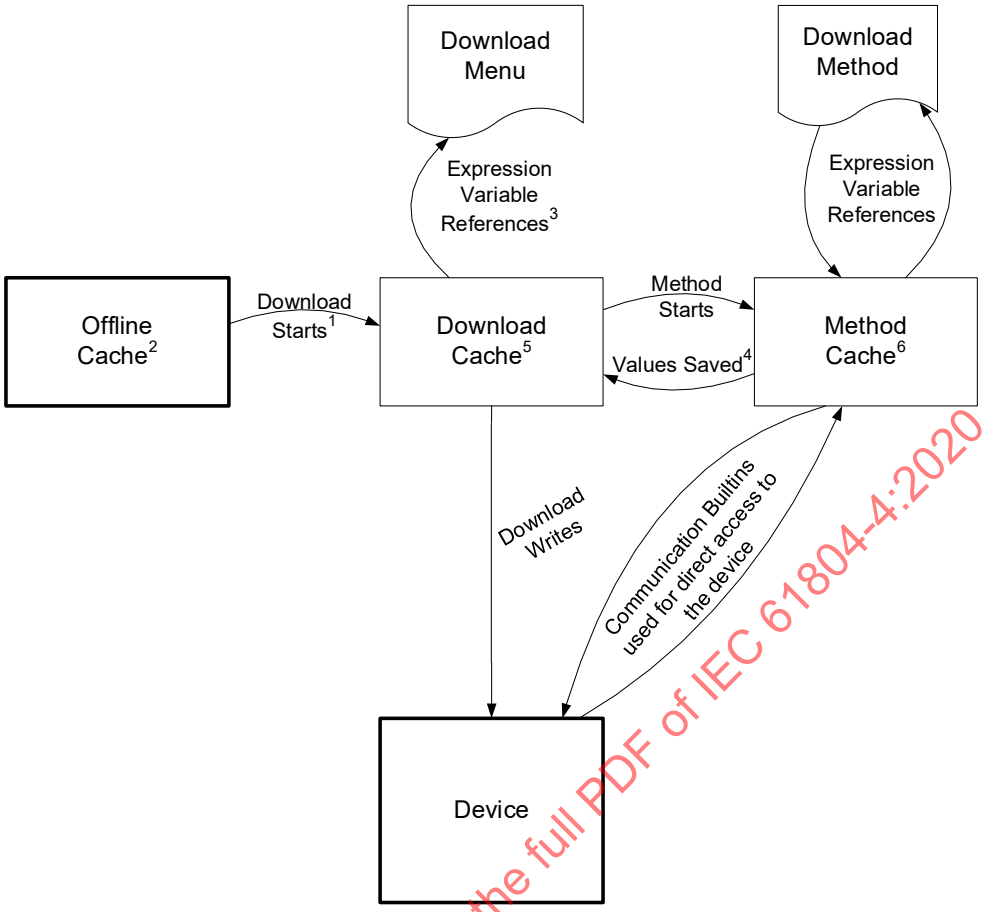
2 Offline cache is continuously synchronized with variable values uploaded from the device.

3 Expressions on the upload menu are evaluated with values from the upload cache.

4 Method cache values are saved back to the upload cache based on rules defined in Table 34.

5 The METHOD cache is used for METHOD, VARIABLE ACTIONS, MENU ACTIONS, etc.

Figure D.1 – Upload caching model



Key

1 Download Cache is initialized from the Offline Data Set.

2 Offline Cache shall not automatically synchronize with the Download Cache.

3 Expressions on the Download Menu are evaluated with values from the Download Cache.

4 Method Cache values are saved back to the Download Cache based on rules defined in Table 34.

5 For HART, values are updated with “Set to nearest value” values from the write response.

6 The METHOD cache is used for METHOD, VARIABLE ACTIONS, MENU ACTIONS, etc.

Figure D.2 – Download caching model

Bibliography

IEC 61804-2:2018, *Function blocks (FB) for process control and electronic device description language (EDDL) – Part 2: Specification of FB concept*

IEC 62769-100[7], *Field device integration (FDI) – Part 100: Profiles – Generic protocols*

IEC 62769-115-2, *Field device integration (FDI) – Part 115-2: Profiles – Modbus-RTU*

IEEE 754, *IEEE Standard for Floating-Point Arithmetic*

ISA-100.11a, *Wireless systems for industrial automation: Process control and related applications*

[7] Under preparation. Stage at the time of publication: IEC AFDIS 62769-100:2020.

SOMMAIRE

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

_____

LES DISPOSITIFS ET LEUR INTÉGRATION DANS LES SYSTÈMES DE L'ENTREPRISE – BLOCS FONCTIONNELS (FB) POUR LES PROCÉDÉS INDUSTRIELS ET LE LANGAGE DE DESCRIPTION ÉLECTRONIQUE DE PRODUIT (EDDL) –

Partie 4: Interprétation EDD

AVANT-PROPOS

1) La Commission Electrotechnique Internationale (IEC) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de l'IEC). L'IEC a pour objet de favoriser la coopération internationale pour toutes les questions de normalisation dans les domaines de l'électricité et de l'électronique. A cet effet, l'IEC – entre autres activités – publie des Normes internationales, des Spécifications techniques, des Rapports techniques, des Spécifications accessibles au public (PAS) et des Guides (ci-après dénommés "Publication(s) de l'IEC"). Leur élaboration est confiée à des comités d'études, aux travaux desquels tout Comité national intéressé par le sujet traité peut participer. Les organisations internationales, gouvernementales et non gouvernementales, en liaison avec l'IEC, participent également aux travaux. L'IEC collabore étroitement avec l'Organisation Internationale de Normalisation (ISO), selon des conditions fixées par accord entre les deux organisations.
2) Les décisions ou accords officiels de l'IEC concernant les questions techniques représentent, dans la mesure du possible, un accord international sur les sujets étudiés, étant donné que les Comités nationaux de l'IEC intéressés sont représentés dans chaque comité d'études.
3) Les Publications de l'IEC se présentent sous la forme de recommandations internationales et sont agréées comme telles par les Comités nationaux de l'IEC. Tous les efforts raisonnables sont entrepris afin que l'IEC s'assure de l'exactitude du contenu technique de ses publications; l'IEC ne peut pas être tenue responsable de l'éventuelle mauvaise utilisation ou interprétation qui en est faite par un quelconque utilisateur final.
4) Dans le but d'encourager l'uniformité internationale, les Comités nationaux de l'IEC s'engagent, dans toute la mesure possible, à appliquer de façon transparente les Publications de l'IEC dans leurs publications nationales et régionales. Toutes divergences entre toutes Publications de l'IEC et toutes publications nationales ou régionales correspondantes doivent être indiquées en termes clairs dans ces dernières.
5) L'IEC elle-même ne fournit aucune attestation de conformité. Des organismes de certification indépendants fournissent des services d'évaluation de conformité et, dans certains secteurs, accèdent aux marques de conformité de l'IEC. L'IEC n'est responsable d'aucun des services effectués par les organismes de certification indépendants.
6) Tous les utilisateurs doivent s'assurer qu'ils sont en possession de la dernière édition de cette publication.
7) Aucune responsabilité ne doit être imputée à l'IEC, à ses administrateurs, employés, auxiliaires ou mandataires, y compris ses experts particuliers et les membres de ses comités d'études et des Comités nationaux de l'IEC, pour tout préjudice causé en cas de dommages corporels et matériels, ou de tout autre dommage de quelque nature que ce soit, directe ou indirecte, ou pour supporter les coûts (y compris les frais de justice) et les dépenses découlant de la publication ou de l'utilisation de cette Publication de l'IEC ou de toute autre Publication de l'IEC, ou au crédit qui lui est accordé.
8) L'attention est attirée sur les références normatives citées dans cette publication. L'utilisation de publications référencées est obligatoire pour une application correcte de la présente publication.
9) L'attention est attirée sur le fait que certains des éléments de la présente Publication de l'IEC peuvent faire l'objet de droits de brevet. L'IEC ne saurait être tenue pour responsable de ne pas avoir identifié de tels droits de brevets et de ne pas avoir signalé leur existence.

La Norme internationale IEC 61804-4 a été établie par le sous-comité 65E: Les dispositifs et leur intégration dans les systèmes de l'entreprise, du comité d'études 65 de l'IEC: Mesure, commande et automation dans les processus industriels.

Cette deuxième édition annule et remplace la première édition parue en 2015. Cette édition constitue une révision technique.

Cette édition a été élaborée en fusionnant le contenu de plusieurs variantes des spécifications de l'EDDL existantes, y compris celles du FieldComm Group (Foundation™ Fieldbus[1], HART®[2]), du PROFIBUS™[3] Nutzerorganisation e.V. (PNO), et de l'ISA100_Wireless™[4] Compliance Institute (ISA100 WCI). Lorsqu'une déviation de profil existe, cela est désormais indiqué dans le contexte où la déviation en question a été identifiée. Par conséquent, le formatage et la numérotation de la présente édition peuvent différer des spécifications individuelles desquelles elle est issue.

Cette édition inclut les modifications techniques majeures suivantes par rapport à l'édition précédente:

- ajout des profils de communication ISA100 et GPE;
- ajout de la description des règles pour la disposition de largeur optimale des colonnes;
- ajout de la description de la concaténation des libellés et des aides;
- ajout de bandes de couleurs sur les diagrammes de type instrument de mesure.

Le texte de cette Norme internationale est issu des documents suivants:

CDV	Rapport de vote
65E/633/CDV	65E/690/RVC

Le rapport de vote indiqué dans le tableau ci-dessus donne toute information sur le vote ayant abouti à l'approbation de cette Norme internationale.

Cette publication a été rédigée selon les Directives ISO/IEC, Partie 2.

Une liste de toutes les parties de la série IEC 61804, publiées sous le titre général *Les dispositifs et leur intégration dans les systèmes de l'entreprise – Blocs fonctionnels (FB) pour les procédés industriels et le langage de description électronique de produit (EDDL)*, peut être consultée sur le site web de l'IEC.

Les futures normes de cette série porteront dorénavant le nouveau titre général cité ci-dessus. Le titre des normes existant déjà dans cette série sera mis à jour lors de la prochaine édition.

1 Foundation™ Fieldbus est l'appellation commerciale du FieldComm Group. Cette information est donnée à l'intention des utilisateurs du présent document et ne signifie nullement que l'IEC approuve ou recommande l'emploi exclusif du produit ainsi désigné. Des produits équivalents peuvent être utilisés s'il est démontré qu'ils conduisent aux mêmes résultats.

2 HART® est une marque déposée du FieldComm Group. Cette information est donnée à l'intention des utilisateurs du présent document et ne signifie nullement que l'IEC approuve ou recommande l'emploi exclusif du produit ainsi désigné. Des produits équivalents peuvent être utilisés s'il est démontré qu'ils conduisent aux mêmes résultats.

3 PROFIBUS et PROFINET sont les appellations commerciales du PROFIBUS Nutzerorganisation e.V. Cette information est donnée à l'intention des utilisateurs du présent document et ne signifie nullement que l'IEC approuve ou recommande l'emploi exclusif du produit ainsi désigné. Des produits équivalents peuvent être utilisés s'il est démontré qu'ils conduisent aux mêmes résultats.

4 ISA100_Wireless™ est l'appellation commerciale de l'ISA100 Wireless Compliance Institute. Cette information est donnée à l'intention des utilisateurs du présent document et ne signifie nullement que l'IEC approuve ou recommande l'emploi exclusif du produit ainsi désigné. Des produits équivalents peuvent être utilisés s'il est démontré qu'ils conduisent aux mêmes résultats.

Le comité a décidé que le contenu de cette publication ne sera pas modifié avant la date de stabilité indiquée sur le site web de l'IEC sous "http://webstore.iec.ch" dans les données relatives à la publication recherchée. A cette date, la publication sera

- reconduite,
- supprimée,
- remplacée par une édition révisée, ou
- amendée.

IMPORTANT – Le logo "*colour inside*" qui se trouve sur la page de couverture de cette publication indique qu'elle contient des couleurs qui sont considérées comme utiles à une bonne compréhension de son contenu. Les utilisateurs devraient, par conséquent, imprimer cette publication en utilisant une imprimante couleur.

INTRODUCTION

La présente partie de l'IEC 61804:

- contient une présentation de l'utilisation du langage EDDL;
- fournit des exemples qui représentent l'utilisation des constructions EDDL;
- montre comment les cas d'utilisation sont réalisés; et
- montre l'interprétation correcte d'une application EDDL pour chaque exemple.

La présente partie de l'IEC 61804 n'est pas une explication EDDL et n'est pas destinée à remplacer la spécification EDDL.

Des instructions sont données pour l'application EDD, qui décrivent les actions qui seront réalisées sans la prescription de la technologie utilisée dans la mise en œuvre de l'hôte. Par exemple, la construction FILE décrit les données sauvegardées par l'application EDD pour le compte de l'EDD. La construction FILE ne spécifie pas la manière dont les données sont sauvegardées. L'application EDD peut utiliser une base de données, un fichier plat ou toute autre mise en œuvre de son choix.

Les fonctions EDDL sont limitées par profil pour chacune des technologies de communication. Les descriptions dans la présente partie de l'IEC 61804 font référence à ces fonctions au sens général et toutes les technologies de communication ne prendront pas en charge toutes les fonctions décrites. Il est fait référence aux définitions de profil de l'IEC 61804-3 pour comprendre les fonctions prises en charge par chaque technologie de communication.

LES DISPOSITIFS ET LEUR INTÉGRATION DANS LES SYSTÈMES DE L'ENTREPRISE – BLOCS FONCTIONNELS (FB) POUR LES PROCÉDÉS INDUSTRIELS ET LE LANGAGE DE DESCRIPTION ÉLECTRONIQUE DE PRODUIT (EDDL) –

Partie 4: Interprétation EDD

1 Domaine d'application

La présente partie de l'IEC 61804 définit l'interprétation EDD pour les applications EDD et les EDD afin d'assurer l'interopérabilité EDD. Le présent document est destiné à garantir que les développeurs d'appareils de terrain utilisent systématiquement les constructions EDDL et que les applications EDD aient la même interprétation des EDD. Il complète la spécification EDDL pour promouvoir l'interopérabilité de l'application EDDL et améliorer la portabilité EDD entre les applications EDDL.

2 Références normatives

Les documents suivants cités dans le texte constituent, pour tout ou partie de leur contenu, des exigences du présent document. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

IEC 61784-1, *Réseaux de communication industriels – Profils – Partie 1: Profils de bus de terrain*

IEC 61784-2, *Réseaux de communication industriels – Profils – Partie 2 : Profils de bus de terrain supplémentaires pour les réseaux en temps réel fondés sur l'ISO/IEC/IEEE 8802-3*

IEC 61804-3, *Les dispositifs et leur intégration dans les systèmes de l'entreprise – Blocs Fonctionnels (FB) pour les procédés industriels et le langage de description électronique de produit (EDDL) – Partie 3: Sémantique et syntaxe EDDL*

IEC 61804-5, *Les dispositifs et leur intégration dans les systèmes de l'entreprise – Blocs Fonctionnels (FB) pour les procédés industriels et le langage de description électronique de produit (EDDL) – Partie 5: Bibliothèque de Builtin EDDL*

IEC 62734, *Réseaux industriels – Réseau de communication sans fil et profils de communication – ISA 100.11a*

IEC 62769-4[5], *Field Device Integration (FDI) – Part 4: FDI Packages* (disponible en anglais seulement)

IEC 62769-7[6], *Field Device Integration (FDI) – Part 7: FDI Communication devices* (disponible en anglais seulement)

---

5 En cours d'élaboration. Stade au moment de la publication: IEC RFDIS 62769-4:2020.

6 En cours d'élaboration. Stade au moment de la publication: IEC RFDIS 62769-7:2020.

3 Termes, définitions, termes abrégés, acronymes et conventions

3.1 Termes généraux et définitions

Pour les besoins du présent document, les termes et définitions donnés dans l'IEC 61804-3 ainsi que les suivants s'appliquent.

L'ISO et l'IEC tiennent à jour des bases de données terminologiques destinées à être utilisées en normalisation, consultables aux adresses suivantes:

- IEC Electropedia: disponible à l'adresse http://www.electropedia.org/
- ISO Online browsing platform: disponible à l'adresse http://www.iso.org/obp

3.1.1
développeur EDD
personne ou équipe qui développe une EDD

3.1.2
conteneur
éléments de l'interface utilisateur qui contiennent d'autres éléments d'interface utilisateur

Note 1 à l'article: Les conteneurs peuvent être des menus, fenêtres, boîtes de dialogue, tableaux, pages, groupes et autres conteneurs.

3.1.3
élément contenu
éléments de l'interface utilisateur qui peuvent être contenus dans des conteneurs

Note 1 à l'article: Les éléments contenus peuvent être des variables, méthodes, graphiques, diagrammes, tableaux, images, textes statiques.

3.1.4
développeur de l'appareil
personne ou équipe qui développe un appareil et une EDD qui décrit l'appareil

3.1.5
portatif
appareil dont la résolution d'affichage limitée restreint l'interface utilisateur d'applications EDD

3.2 Termes et définitions relatifs aux appareils modulaires

3.2.1
canal
connexion à un processus qui est mesuré ou contrôlé

3.2.2
composant
logiciel ou matériel contenu dans le concept d'appareil modulaire

Note 1 à l'article: Un composant ne peut pas fonctionner séparément d'un appareil modulaire hôte. Un composant peut prendre en charge un ou plusieurs types d'appareils modulaires.

3.2.3
interface
déclarations de base de constructions de base

Note 1 à l'article: Une interface définit toutes les parties publiques que les composants peuvent utiliser.

3.2.4
fenêtre modale
fenêtre enfant qui exige que les utilisateurs interagissent avec elle pour pouvoir revenir à l'exploitation de l'application parente et éviter ainsi de bloquer le flux de travail dans la fenêtre principale de l'application

3.2.5
appareil modulaire
appareil qui peut contenir une gamme de composants logiciels et/ou matériels divers

3.3 Abréviations et acronymes

CP	Communication Profile (Profil de communication)
CPF	Communication Profile Family (Famille de profils de communication)
CS	Communication Server (Serveur de communication)
EDD	Electronic Device Description (Description électronique de produit)
EDDL	Electronic Device Description Language (Langage de description électronique de produit)
FDI	Field Device Integration (Intégration des appareils de terrain)
FF	Foundation Fieldbus
GPE	Generic Protocol Extension (Extension de protocole générique)
HART	Highway Addressable Remote Transducer (Transducteur de canal adressable distant)
ISA100	ISA100_Wireless™
PB	PROFIBUS
PC	Personal computer (Ordinateur personnel)
PI	PROFIBUS et PROFINET International
PI PROFILE PA	Profil spécifique PI pour les appareils de terrain d'automatisation de processus
PN	PROFINET

3.4 Conventions

Il existe certaines différences d'utilisation et d'interprétation d'EDDL en fonction du réseau de communication utilisé et du modèle de l'appareil. L'Annexe C inclut une description de chacun des profils EDDL. Les différents réseaux de communication utilisés dans la présente partie de l'IEC 61804 sont:

- HART (conformément à l'IEC 61784-1 CPF9)
- FOUNDATION fieldbus (conformément à l'IEC 61784-1 CPF1)
- PROFIBUS (conformément à l'IEC 61784-1 CPF3)
- PROFINET (conformément à l'IEC 61784-2 CPF3)
- CS (Serveur de communication)
  - IEC 62769-4 Spécification technique FDI: Partie 4: Paquetages FDI
  - IEC 62769-7 Spécification technique FDI: Partie 7: Appareils de communication
- ISA100_Wireless (conformément à l'IEC 62734)
- GPE (Extension de protocole générique)
  - IEC 62769-4 Spécification technique FDI: Partie 4: Paquetages FDI

– IEC 62769-7 Spécification technique FDI: Partie 7: Appareils de communication.

Les exemples d'EDD donnés dans le présent document ne montrent que certaines parties de la mise en œuvre de l'EDD et sont incomplets.

Les mots-clés EDDL sont écrits en majuscules. Si plusieurs éléments de construction de base sont impliqués, le mot-clé est écrit en majuscules suivies d'un "s" minuscule, par exemple VARIABLEs.

Builtin écrit avec une majuscule fait référence à des fonctions spécifiées dans l'IEC 61804-5.

EXEMPLE Builtin MenuDisplay fait référence au Builtin appelé MenuDisplay spécifié dans l'IEC 61804-5.

4 Description de l'interface utilisateur EDDL

4.1 Vue d'ensemble

La plupart des applications EDD peuvent être caractérisées comme applications PC ou portatives. Au vu de la taille relativement petite d'un appareil portatif, les applications portatives ne peuvent afficher qu'un petit nombre d'informations à la fois. En revanche, les applications PC peuvent proposer une interface utilisateur bien plus intéressante, en grande partie du fait de leur taille d'écran supérieure.

Pour prendre en charge les capacités des applications PC, la construction MENU a été étendue dans l'IEC 61804-3 par rapport aux définitions précédentes données dans l'IEC 61804-2:2004. Etant donné les différences au niveau des interfaces utilisateurs des applications PC et des applications portatives, de nombreux appareils définiront par logique deux hiérarchies de MENU: la première pour les applications portatives et la seconde pour les applications PC. Certains MENUs peuvent être utilisés dans les deux hiérarchies. Il n'est donc pas nécessaire de spécifier la hiérarchie entière en double.

Différentes structures de menu sont possibles en fonction des différentes classes d'applications. Le présent document doit être utilisé pour créer des structures de menu dans une EDD qui soient interprétées de manière non équivoque par les applications. Afin d'assurer une l'interopérabilité au travers des applications, le présent document doit être appliqué. L'Annexe B liste les identificateurs prédéfinis qui peuvent être utilisés par les applications EDD pour accéder aux informations normalisées.

4.2 Conventions de menus pour les applications portatives

Les applications EDD utilisent des menus spécifiques dans les EDD pour afficher l'interface utilisateur de l'appareil (voir Tableau 1). Les éléments de l'interface utilisateur peuvent en outre être décrits en même temps pour les appareils portatifs et pour les PCs. Pour les appareils portatifs, les chaînes peuvent avoir, en plus d'un code de langue, un code zz spécifique à leur pays afin de spécifier les chaînes plus courtes ou les images de moindre résolution (voir l'IEC 61804-3). Afin de s'adapter à un espace d'affichage limité, une application portative peut restituer tous les menus en utilisant le STYLE TABLE. Les applications portatives doivent toujours restituer les références aux images dans les MENUs de STYLE TABLE sous forme de lien hypertexte.

Tableau 1 – Liste des identificateurs de menu racine définis pour les appareils portatifs

Identificateur de menu	STYLE par défaut	Description succincte
root_menu	TABLE	Menu racine portatif pour les appareils HART. Il est obligatoire pour l'ensemble des EDD HART.
Menu_Top*	TABLE	Préfixe du menu racine hérité pour les BLOCK_A MENU_ITEMs pour les appareils FOUNDATION fieldbus et ISA100_Wireless, par exemple Menu_Top_TB.
offline_root_menu	TABLE	Configuration hors ligne. Les appareils portatifs peuvent éventuellement prendre en charge ce menu.
hh_process_variables_root_menu	TABLE	Affichage des variables de processus pour les appareils FOUNDATION fieldbus et HART.
hh_device_root_menu	TABLE	Affichage des caractéristiques de l'appareil pour configuration des appareils FOUNDATION fieldbus et HART.
hh_diagnostic_root_menu	TABLE	Affichage des diagnostics pour les appareils FOUNDATION fieldbus et HART.

4.3 Conventions de menus pour les applications PC

4.3.1 Vue d'ensemble

Les applications EDD utilisent des menus spécifiques dans les EDD pour afficher l'interface utilisateur de l'appareil. De tels menus sont définis pour le diagnostic, les variables de processus, les caractéristiques des appareils et la configuration hors ligne. Le Tableau 2 définit les identificateurs des différents menus racine avec les STYLEs par défaut. Le STYLE par défaut est utilisé par l'application EDD si l'attribut STYLE n'est pas défini dans le MENU. La colonne "Utilisation" du Tableau 2 définit comment l'application EDD possède un accès en ligne et hors ligne aux paramètres de l'appareil (voir Article 8).

Tableau 2 – Liste des identificateurs de menu racine définis pour les appareils sur PC

Identificateur de MENU	STYLE par défaut	Utilisation	Description succincte
device_root_menu	MENU	En ligne	Affichage des caractéristiques de l'appareil pour configuration
diagnostic_root_menu	MENU	En ligne	Affichage des diagnostics
maintenance_root_menu	MENU	En ligne	Affichage des caractéristiques de maintenance
offline_root_menu	TABLE	Hors ligne	Configuration hors ligne
process_variables_root_menu	MENU	En ligne	Affichage des variables de processus

Pour les EDD HART, l'application EDD doit afficher le menu offline_root_menu avec un STYLE WINDOW par défaut si STYLE n'est pas défini.

Au moyen de sous-menus, PROFIBUS et PROFINET permettent de définir différents paramètres d'accès, par la définition d'ACCESS ONLINE ou d'ACCESS OFFLINE.

Il convient que les EDD contiennent les menus racine en ligne du Tableau 2. Les menus racine en ligne sont facultatifs pour PROFIBUS, PROFINET et FOUNDATION fieldbus. Le menu racine hors ligne est facultatif pour FOUNDATION fieldbus et ISA100_Wireless. Les EDD HART doivent avoir root_menu en plus de tous les menus racine du Tableau 2. GPE doit avoir au moins un menu en ligne. GPE doit avoir le menu offline_root_menu. CS doit avoir au moins un menu en ligne.

4.3.2 Menus racine en ligne

4.3.2.1 Généralités

Si aucun des menus racine en ligne n'existe, des points d'entrée spécifiques pour le profil de communication doivent être utilisés (voir Tableau 3).

Tableau 3 – Alternatives de redémarrage pour les menus racine en ligne

Profil de communication	Description des alternatives de redémarrage
FF, ISA100	Menus déclarés dans l'attribut BLOCK_A MENU_ITEMS, par exemple device_root_menu_aiblock. Si les menus du bloc ne sont pas définis, un MENU de STYLE TABLE doit être généré à partir de l'attribut BLOCK_A PARAMETERS.
HART	root_menu, dans le STYLE TABLE.
PB, PN	Menu_Main_Specialist s'il existe ou Menu_Main_Maintenance. Ces menus incluent des sous-menus en ligne et hors ligne. Ils sont restitués dans le STYLE MENU.

4.3.2.2 Menu racine de diagnostic

Le menu diagnostic_root_menu comprend des affichages qui présentent l'état de l'appareil et des informations détaillées sur le diagnostic, et peut comprendre des affichages graphiques qui indiquent, par exemple, une signature de vanne.

4.3.2.3 Menu racine de variables de processus

Le menu process_variables_root_menu comprend des affichages qui présentent les mesures et points de consigne du processus avec leur qualité et des informations importantes pour les opérateurs de processus, comme les plages.

4.3.2.4 Menu racine d'appareil

Le menu device_root_menu intègre les caractéristiques d'un appareil. Ces caractéristiques peuvent être divisées entre celles qui sont relatives au processus et celles qui sont spécifiques à l'appareil. Cette structure n'est pas exigée si le nombre de caractéristiques est trop petit pour être partagé. Les sous-menus qui représentent une structure sont admis dans le menu des caractéristiques de l'appareil. En cas de création de sous-menus, les menus qui se trouvent en dessous peuvent être divisés entre caractéristiques relatives au processus et caractéristiques spécifiques à l'appareil.

4.3.2.5 Menu racine de maintenance

Il convient que le menu maintenance_root_menu contienne des fonctionnalités permettant la maintenance de l'appareil pendant la phase d'exécution et l'affichage, par exemple, d'informations relatives à la dernière inspection de maintenance.

4.3.3 Menu racine hors ligne

Le menu offline_root_menu est une hiérarchie de menus comportant, par exemple, des éléments de données et des méthodes de configuration hors ligne. Il contient, en particulier, l'ensemble des paramètres de l'appareil spécifiques aux applications et peut également contenir d'importantes variables en lecture seule et inscriptibles. Pour plus d'informations concernant les paramètres spécifiques aux applications, voir 9.3. Le menu peut comporter des méthodes hors ligne, par exemple des assistants de configuration.

Si le menu racine hors ligne n'existe pas, un point d'entrée spécifique pour le profil de communication doit être utilisé (voir Tableau 4).

Tableau 4 – Alternatives de redémarrage pour les menus racine hors ligne

Profil de communication	Description des alternatives de redémarrage
FF, ISA100	BLOCK_A PARAMETERS
HART	MENU upload_variables
PB, PN	Table_Main_Specialist s'il existe ou Table_Main_Maintenance

4.3.4 Exemple de structure de menu EDD

L'exemple d'EDD de la Figure 1 comporte des menus supplémentaires qui peuvent être ajoutés à une EDD pour des applications PC. Ces menus sont ajoutés aux menus existants pour les applications. Cet exemple est spécifique à un appareil avec une interface conforme à HART. Les exemples pour les appareils avec une interface conforme à FOUNDATION fieldbus, ISA100_Wireless, PROFIBUS, PROFINET, au profil CS et au profil GPE ressembleraient beaucoup à celui-ci.

```
MENU diagnostic_root_menu
{
    LABEL "Diagnostics";
    STYLE MENU;                         /* non exigé pour définir le STYLE dans ce cas, */
                                        /* en raison du STYLE du menu racine par défaut */
    ITEMS
    {
        status_window,                  /* menu: style=window */
        self_test                       /* méthode */
    }
}

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
    {
         /* élidé */
    }
}

MENU maintenance_root_menu
{
    LABEL "Maintenance";
    STYLE MENU;                         /* non exigé pour définir le STYLE dans ce cas, */
                                        /* en raison du STYLE du menu racine par défaut */

    ITEMS
    {
        device_mode_dialog,                 /* menu: style=dialog */
        teach_in                            /* méthode */
    }
}
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
    {
        /* élidé */
    }
}

MENU process_variables_root_menu
{
    LABEL "Process Variables";
    STYLE MENU;                     /* non exigé pour définir le STYLE dans ce cas, */
                                    /* en raison du STYLE du menu racine par défaut */

    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              /* diagramme */
    }
}

CHART pressure_chart
{
    /* élidé */
}

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

CHART temperature_chart
{
    /* élidé */
}

MENU device_root_menu
{
    LABEL "Device";
    STYLE MENU;                     /* non exigé pour définir le STYLE dans ce cas, */
                                    /* en raison du STYLE du menu racine par défaut */

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

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                  /* méthode */
    }
}

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

Figure 1 – Exemple de menus racine EDD

4.3.5 Interface utilisateur

4.3.5.1 Diagnostics

La Figure 2 représente un exemple d'application EDD pour les diagnostics.

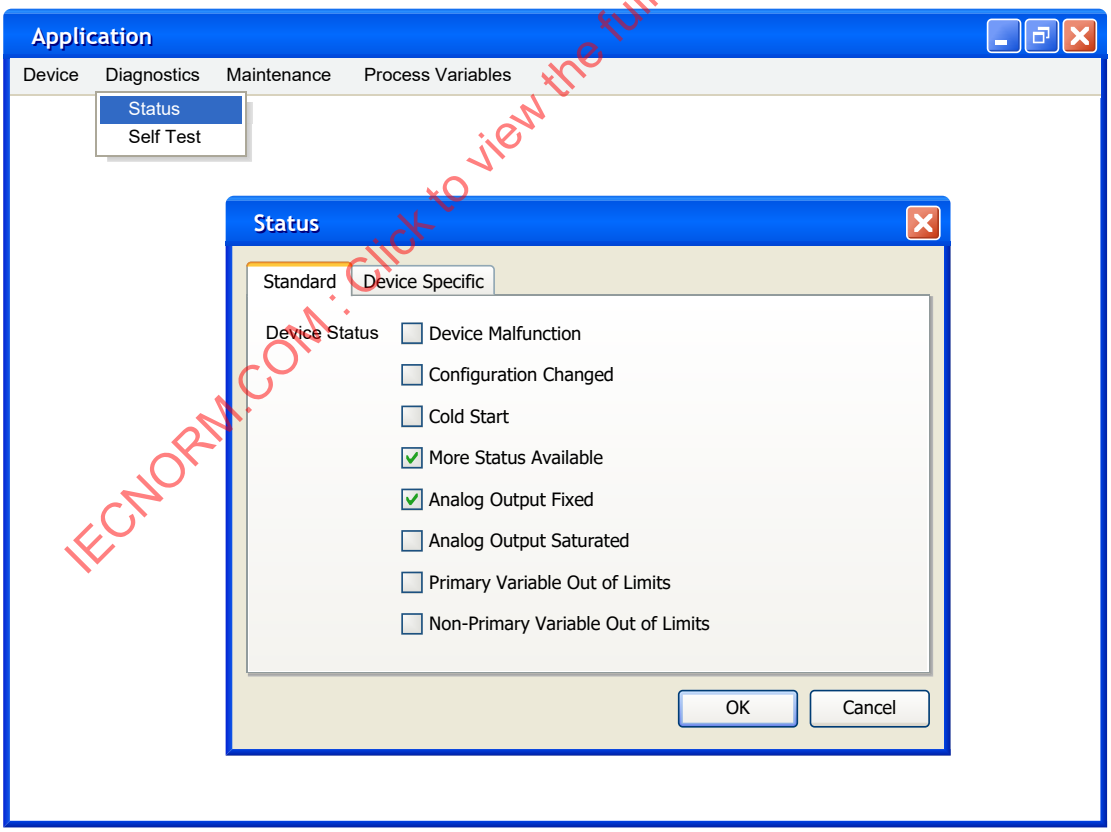


Figure 2 – Exemple d'application EDD pour les diagnostics

4.3.5.2 Variables de processus

La Figure 3 et la Figure 4 représentent des exemples d'applications EDD pour les variables de processus.

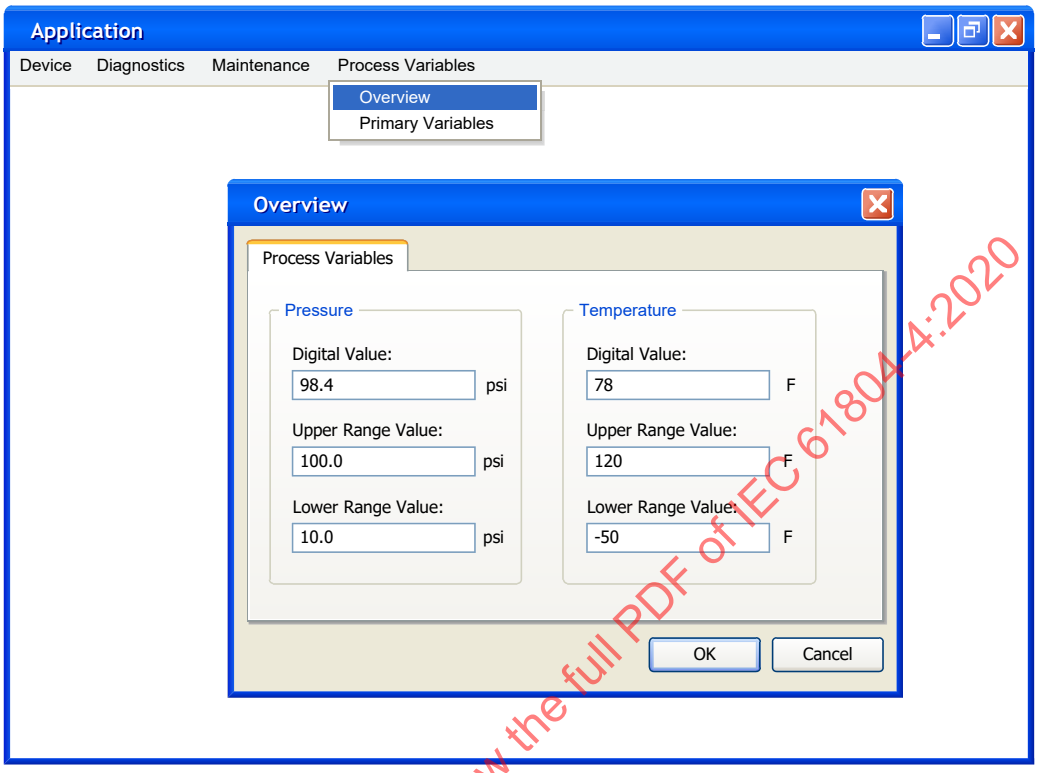


Figure 3 – Exemple d'application EDD pour les variables de processus

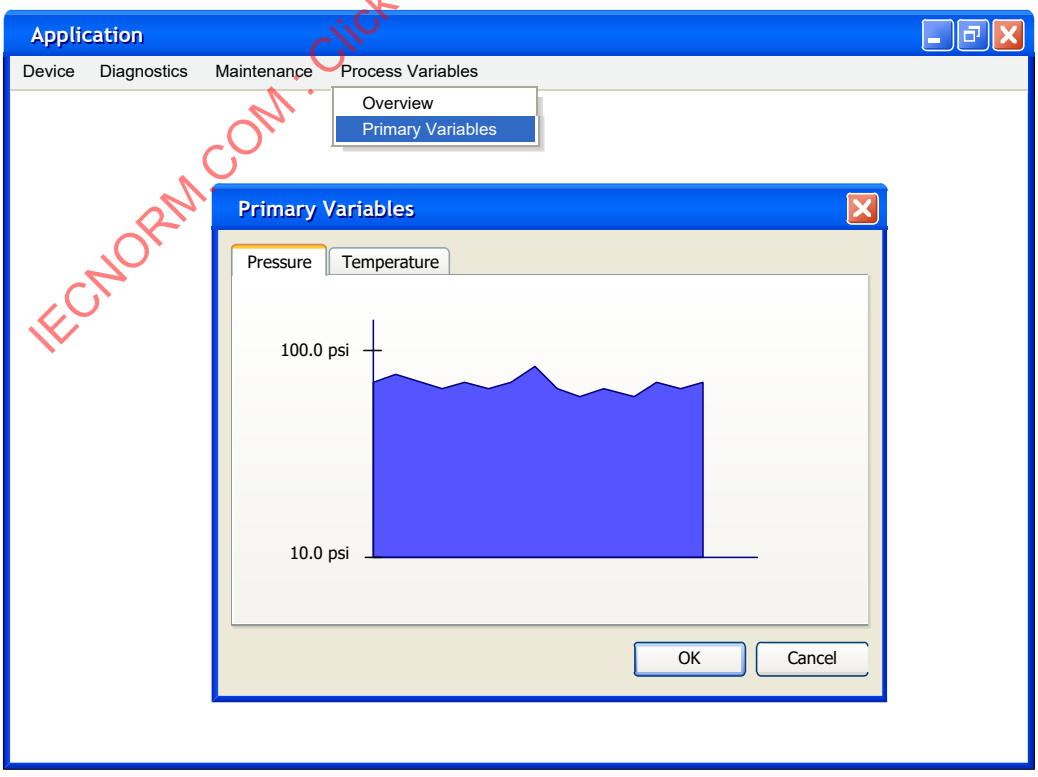


Figure 4 – Exemple d'application EDD pour les variables principales

4.3.5.3 Caractéristiques de l'appareil

La Figure 5, la Figure 6 et la Figure 7 représentent des exemples d'applications EDD pour les caractéristiques de l'appareil.

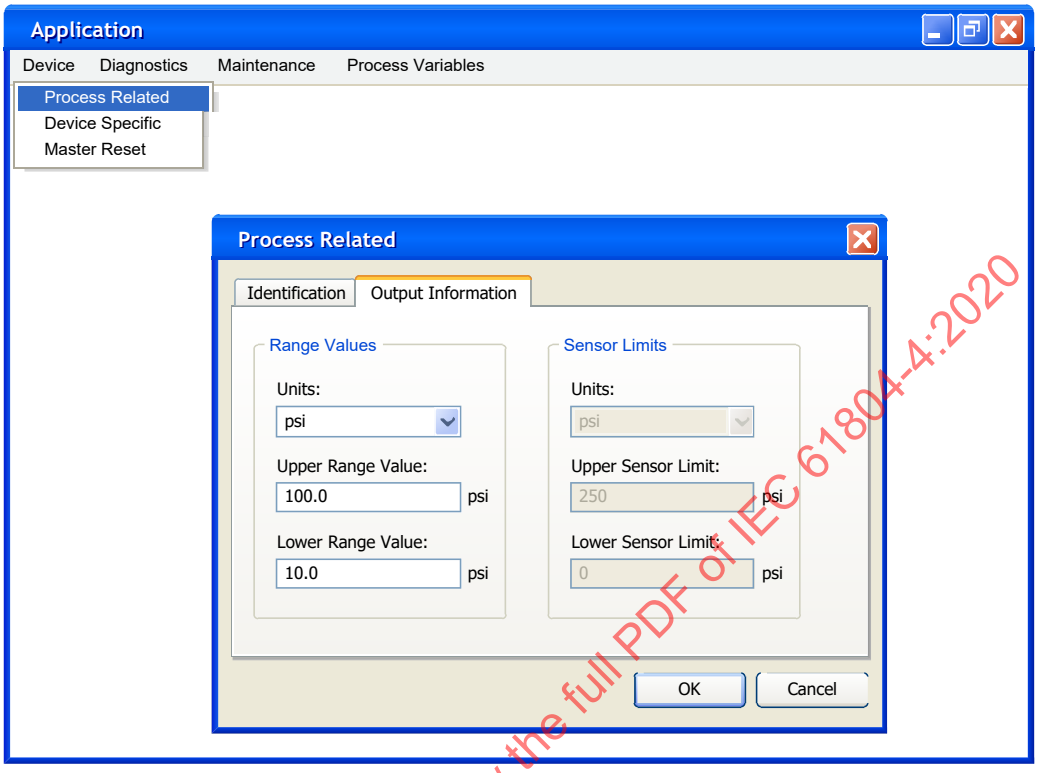


Figure 5 – Exemple d'application EDD pour les caractéristiques de l'appareil relatives au processus

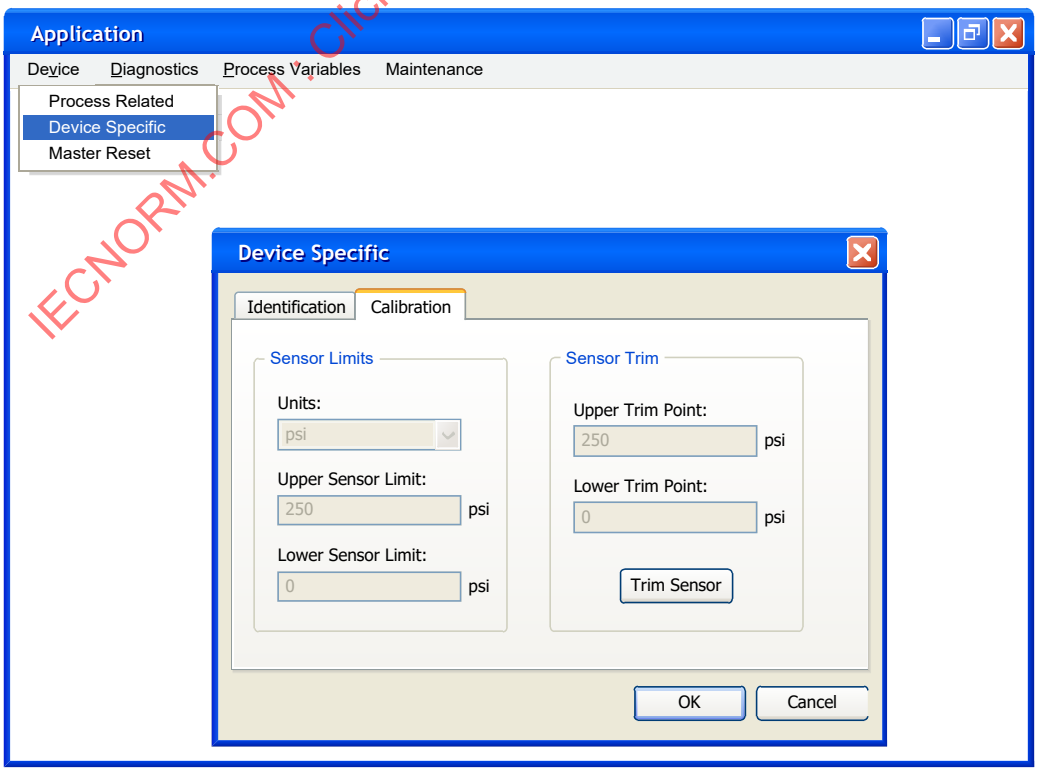


Figure 6 – Exemple d'application EDD pour les caractéristiques de l'appareil

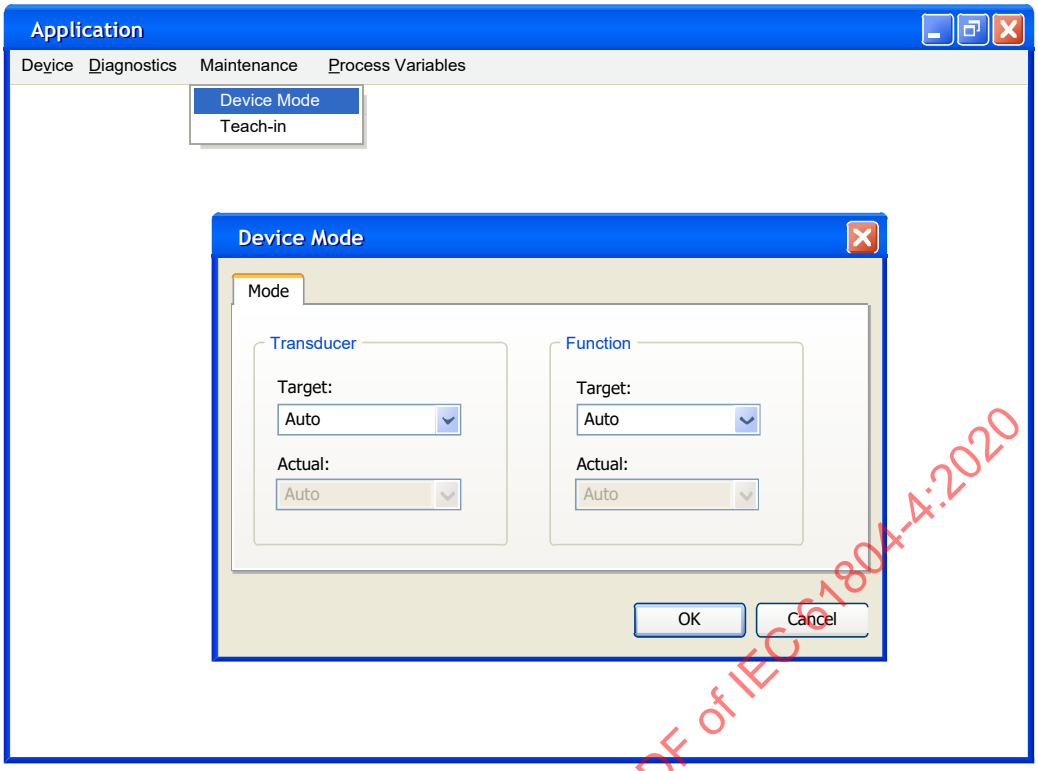


Figure 7 – Exemple d'application EDD pour les caractéristiques de maintenance

4.4 Concaténation des libellés pour les références de variables indirectes

4.4.1 Généralités

Pour les références de variables indirectes seulement, le LABEL affiché doit être une concaténation des chaînes trouvées dans les conteneurs d'éléments EDD par le biais desquels la VARIABLE était référencée. Cela se produit, par exemple, lorsqu'une VARIABLE est référencée par le biais d'un COLLECTION MEMBER ou d'un ARRAY ELEMENT. PV.DEVICE_VARIABLE.DIGITAL_VALUE est, par exemple, une référence indirecte à une VARIABLE par le biais à la fois des matrices et des collections. Une référence est définie comme suit:

```
reference:= [ identifier | reference [ expression ] |
              reference . identifier ]
```

Pour spécifier la concaténation d'un Label, les références de variables sont classées comme étant directes, simples ou complexes.

Une "référence directe" est:

```
direct-reference:= identifier
```

La concaténation de chaîne n'est pas applicable aux références directes.

Une "référence simple" est:

```
simple-reference:= [ identifier [ expression ] | identifier . identifier ]
```

Et une "référence complexe" est:

```
complex-reference:= [ simple-reference . identifier |
                      simple-reference [ expression ] |
                      complex-reference . identifier |
                      complex-reference [ expression ] ]
```

La concaténation d'un Label est uniquement applicable aux références de variables (indirectes) simples et complexes. Les références de variables directes n'ont pas de Labels concaténés.

Dans les paragraphes ci-après, tout libellé ou description contenant une chaîne vide ("") doit être traité comme s'il n'existait pas.

4.4.2 Références de variables simples

Pour les références de variables simples, le Label et la Description de la référence de variable simple sont concaténés ensemble. Voir le récapitulatif du Tableau 5 et du Tableau 6.

Tableau 5 – Récapitulatif de la règle des libellés pour les références de variables simples

Type de référence	Libellé + description existent	Seul le libellé existe	Seule la description existe	Ni le libellé ni la description n'existe
FILE REFERENCE_ARRAY COLLECTION	Le libellé doit correspondre à la concaténation du Label et de la Description du terme.	Le libellé doit être le Label du terme.	Le libellé doit être la Description du terme.	Même Label en référence de variable directe.
RECORD	Le libellé doit être la Description du terme.	Le libellé doit être le Label du terme.	Le libellé doit être la Description du terme.	Même Label en référence de variable directe.

Tableau 6 – Récapitulatif de la règle des libellés pour les références de variables simples

Type de référence	Le libellé existe	
	Oui	Non
LIST VALUE_ARRAY	Le libellé doit être le Label "[n]", où n est la valeur de l'index.	Le libellé doit être "[n]", où n est la valeur de l'index.

4.4.3 Références de variables complexes

Le LABEL d'une référence de variable complexe est construit en concaténant les chaînes Prefix, Body et Suffix ensemble.

Chaîne Prefix

La chaîne Prefix LABEL est générée sur la base du premier terme de la référence de variable complexe. Dans le cas d'un fichier, d'une matrice de référence, d'un bloc, d'un enregistrement ou d'une référence de collection, la première chaîne Description non nulle est utilisée. Voir les règles de Prefix récapitulées dans le Tableau 7 et le Tableau 8.

Noter que si un FILE, une REFERENCE_ARRAY, un BLOCK ou une référence de COLLECTION est le premier terme de la référence de variable complexe et n'a pas de chaîne Description correspondante, la recherche d'une chaîne Prefix continue avec le terme suivant dans la référence de variable complexe. Ce processus est répété jusqu'à ce que la chaîne Prefix soit générée ou que le dernier terme de la référence de variable complexe soit rencontré. Si la chaîne Prefix est générée, le processus continue avec la construction des chaînes Body et Suffix.

Si le dernier terme est rencontré sans qu'aucune chaîne Prefix ne soit générée, alors les règles des références de variables simples sont utilisées pour générer le LABEL en utilisant le dernier terme de la référence de variable complexe.

Tableau 7 – Récapitulatif de la règle de préfixe pour les références de variables complexes

Type de référence	La description existe	
	Oui	Non
FILE REFERENCE_ARRAY COLLECTION RECORD BLOCK_A	Le préfixe est Description.	Ignorer ce terme. Continuer jusqu'au prochain terme de la référence de variable complexe afin de localiser la chaîne Prefix.

Tableau 8 – Récapitulatif de la règle de préfixe pour les références de variables complexes

Type de référence	Le libellé existe	
	Oui	Non
LIST VALUE_ARRAY	Le préfixe doit être le Label "[n]", où n est la valeur de l'index.	Le préfixe doit être "[n]", où n est la valeur de l'index.

Chaîne Body

La chaîne Body est une concaténation séquentielle de chaînes générées à partir des termes intermédiaires de la référence de variable complexe. Voir le Tableau 9 et le Tableau 10 pour un récapitulatif des règles de construction de la chaîne Body.

La génération de la chaîne Body s'arrête lorsque le dernier terme de la référence de variable complexe est atteint. La chaîne Body doit être la chaîne nulle (autrement dit "") si le dernier terme est atteint avant création d'une chaîne Body. Le dernier terme de la référence de variable complexe doit être utilisé pour générer le Suffix.

Tableau 9 – Récapitulatif de la règle de Body pour les références de variables complexes

Type de référence	
FILE COLLECTION RECORD	Rien n'est concaténé avec la chaîne Body.
LIST VALUE_ARRAY	[n] est concaténé avec la chaîne Body, où n est la valeur de l'index trouvée dans la référence.

Tableau 10 – Récapitulatif de la règle de Body pour les références de variables complexes

Type de référence	Le préfixe est issu d'un FILE, d'une COLLECTION ou d'une REFERENCE_ARRAY	
	Oui	Non
REFERENCE_ARRAY	Rien n'est concaténé avec la chaîne Body.	[n] est concaténé avec la chaîne Body, où n est l'index.

Chaîne Suffix

La chaîne Suffix est générée à partir du dernier terme de la référence de variable complexe. Voir le Tableau 11 et le Tableau 12 pour un récapitulatif des règles de construction de la chaîne Suffix.

Tableau 11 – Récapitulatif de la règle de suffixe pour les références de variables complexes

Type de référence	
LIST VALUE_ARRAY	[n] est le Suffix, où n est la valeur de l'index.

Tableau 12 – Récapitulatif de la règle de suffixe pour les références de variables complexes

Type de référence	La description existe	
	Oui	Non
FILE REFERENCE_ARRAY COLLECTION RECORD	Le suffixe est Description.	Aucun Suffix (c'est-à-dire que la chaîne est vide).

4.5 Concaténation des aides

4.5.1 Généralités

La chaîne Help d'une référence de variable simple ou complexe est générée d'une manière similaire à celle utilisée pour la chaîne Label et varie selon le type de la référence de variable (simple ou complexe).

Dans les paragraphes ci-après, toute aide contenant une chaîne vide ("") doit être traitée comme si elle n'existait pas.

4.5.2 Références de variables simples

Pour les références de variables simples, la chaîne d'aide est générée en utilisant les règles énoncées dans le Tableau 13 et le Tableau 14.

Tableau 13 – Récapitulatif de la règle des aides pour les références de variables simples

Type de référence	L'aide du conteneur + du membre/de l'élément existent	Seule l'aide du conteneur existe	Seule l'aide du membre/de l'élément existe	Aucune aide n'existe
FILE REFERENCE_ARRAY COLLECTION RECORD	L'aide doit être la concaténation des deux chaînes Help.	L'aide doit être le composant Help du Container.	L'aide doit être le composant Help du Member/Element.	L'aide doit être la même qu'avec une référence de variable directe.

Tableau 14 – Récapitulatif de la règle des aides pour les références de variables simples

Type de référence	L'aide du conteneur existe	
	Oui	Non
LIST VALUE_ARRAY	L'aide doit être le composant Help du Container.	L'aide doit être une chaîne vide.

4.5.3 Références de variables complexes

Pour une référence de variable complexe, en général, il est possible qu'il existe un Prefix et un Suffix de chaîne Help. Contrairement à la concaténation de Label, il n'y a pas de Body de chaîne Help. La chaîne Help est essentiellement le premier composant Help trouvé concaténé avec le composant Help du dernier terme de la référence de variable complexe. Si le Prefix et le Suffix Help sont des chaînes vides, alors la chaîne Help doit être une chaîne vide.

Le Prefix Help est généré sur la base du premier terme de la référence de variable complexe. Dans le cas d'un FILE, d'une REFERENCE_ARRAY ou d'une référence de COLLECTION, le premier composant Help d'un Member/Element non vide est utilisé. Voir les règles de Prefix récapitulées dans le Tableau 15 et le Tableau 16.

Si le dernier terme est rencontré sans qu'aucun Prefix ne soit généré, alors les règles des références de variables simples sont utilisées pour générer le composant Help sur la base du dernier terme de la référence de variable complexe.

Tableau 15 – Récapitulatif de la règle de préfixe d'aide pour les références de variables complexes

Type de référence	
LIST VALUE_ARRAY	Le préfixe est une chaîne Help issue de la liste ou de la matrice de valeurs (éventuellement vide)

Tableau 16 – Récapitulatif de la règle de préfixe d'aide pour les références de variables complexes

Type de référence	L'aide du membre/de l'élément existe	
	Oui	Non
FILE REFERENCE_ARRAY COLLECTION RECORD	Le Prefix d'aide est la chaîne Help du Member/Element.	Ignorer cette référence. Continuer jusqu'à la référence suivante afin de localiser la chaîne Prefix du composant HELP.

Le suffixe Help est généré à partir du dernier terme de la référence de variable complexe (voir le Tableau 17 et le Tableau 18).

Tableau 17 – Récapitulatif de la règle de suffixe d'aide pour les références de variables complexes

Type de référence	
LIST VALUE_ARRAY	Le suffixe est une chaîne Help issue de la VARIABLE référencée (éventuellement vide).

Tableau 18 – Récapitulatif de la règle de suffixe d'aide pour les références de variables complexes

Type de référence	L'aide du membre/de l'élément existe	
	Oui	Non
FILE REFERENCE_ARRAY COLLECTION RECORD	Le Suffix d'aide est la chaîne Help du Member/Element.	Aucun Suffix (c'est-à-dire que la chaîne est vide).

4.6 Conteneurs et éléments contenus

4.6.1 Vue d'ensemble

Les extensions de l'interface utilisateur sont fondées sur un modèle simple d'interface utilisateur. Le modèle se décompose en deux concepts:

- les conteneurs; et
- les éléments contenus.

Les conteneurs sont appelés ainsi car ils contiennent d'autres éléments d'interface utilisateur. Les conteneurs peuvent être des menus, fenêtres, boîtes de dialogue, tableaux, pages, groupes et autres conteneurs. Les conteneurs correspondent à un MENU. Ils se distinguent les uns des autres au moyen de l'attribut STYLE. Cet attribut STYLE indique la manière dont le MENU est affiché.

Les éléments contenus peuvent être des variables, méthodes, affichages d'édition, graphiques, diagrammes, images, textes statiques, entre autres.

4.6.2 STYLE admis et par défaut

Le Tableau 19 définit les éléments de l'interface utilisateur admis dans les conteneurs; les STYLEs de substitution et par défaut ne sont pas définis. Si le style d'un menu n'est pas défini, un style par défaut peut être utilisé en fonction de l'emplacement du menu conteneur et du contenu du menu.

Règles générales:

Si l'ACCESS en ligne ou hors ligne d'un MENU contenu est différent de l'accès du conteneur, l'application EDD doit utiliser le STYLE WINDOW dans une fenêtre; sinon, elle doit utiliser le STYLE DIALOG.

Tableau 19 – Eléments contenus admis et STYLES par défaut

Conteneur (Parent)	**Eléments contenus admis (Eléments enfants)**															**STYLE par défaut des éléments contenus (utilisé si le STYLE n'est pas défini ou en cas de conflits)**
	MENU	DIALOG, WINDOW	PAGE	GROUP	TABLE	EDIT_DISPLAY	VARIABLE	COLLECTION, RECORD	ARRAY, LIST	CHART, GRAPH, GRID	IMAGE	texte statique	METHOD	PLUGIN	COLUMNBREAK, ROWBREAK	SEPARATOR	
MENU	✓	1	X	X	10	✓	8	D	8	2	2	✓	✓	✓	6	6	**Profils de communication: CS, FF, GPE, ISA100, PB, PN** MENU si le menu ne comprend aucun élément de données, sinon DIALOG. **Profils de communication: HART** MENU
DIALOG, WINDOW	✓	1	✓	✓	7	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	5	PAGE
PAGE	✓	1	X	✓	7	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	5	GROUP
GROUP	✓	1	X	✓	7	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	5	GROUP si le parent du GROUP parent n'est pas dans le STYLE GROUP. DIALOG devient un bouton ou un lien hypertexte si le parent du GROUP parent est dans le STYLE GROUP.
TABLE	X	9	X	X	✓	✓	3	D	✓	2	3	✓	✓	✓	6	6	TABLE

Légende

✓ Doit être restitué défini pour l'élément contenu; par exemple, pour un "MENU", conformément à son attribut STYLE.

X Non autorisé. Le STYLE par défaut doit être utilisé pour résoudre ce conflit (voir la colonne "STYLE par défaut des éléments contenus").

D Le STYLE par défaut doit être utilisé (voir la colonne "STYLE par défaut des éléments contenus").

1 Doit être restitué comme bouton ou lien hypertexte en utilisant le LABEL. L'élément doit être présenté lorsque le bouton ou lien hypertexte est sélectionné.

2 Doit être restitué comme bouton ou lien hypertexte en utilisant le LABEL. L'élément doit être présenté dans une boîte de dialogue (c'est-à-dire, une fenêtre modale) lorsque le bouton ou lien hypertexte est sélectionné.

3 Doit être restitué comme bouton ou lien hypertexte en utilisant le LABEL, ou en ligne. S'il n'est pas restitué en ligne, alors l'élément doit être présenté dans une boîte de dialogue (c'est-à-dire, une fenêtre modale) lorsque le bouton ou lien hypertexte est sélectionné.

4 Doit être restitué comme bouton ou lien hypertexte en utilisant le LABEL. L'élément doit être présenté dans une fenêtre non modale lorsque le bouton ou lien hypertexte est sélectionné.

5 Doit être interprété comme un COLUMNBREAK.

6 Doit être restitué comme une ligne ou ignoré.

7 **Profils de communication: CS, FF, GPE, ISA100, PB, PN**

Doit être restitué conformément à la note "✓".

Profils de communication: HART

Doit être restitué conformément à la note "**X**".

8 **Profils de communication: CS, FF, GPE, ISA100, PB, PN**

Doit être restitué conformément à la note "**2**".

Profils de communication: HART

Doit être ignoré et non présenté.

9 **Profils de communication: CS, FF, GPE, ISA100, PB, PN**

Doit être restitué conformément à la note "**1**".

Profils de communication: HART

Doit être restitué conformément à la note "**X**".

10 **Profils de communication: CS, FF, GPE, ISA100, PB, PN**

Doit être restitué conformément à la note "**4**".

Profils de communication: HART

Doit être restitué conformément à la note "**D**" si subordonné à DIALOG ou WINDOW, sinon, conformément à la note "**4**".

4.6.3 Conteneurs

4.6.3.1 Menu

Profils de communication: CS, FF, GPE, ISA100, PB, PN

Un menu du STYLE MENU se présente sous la forme d'un menu déroulant, menu contextuel ou d'une barre de navigation.

Profils de communication: HART

Un menu du STYLE MENU se présente sous la forme d'un menu déroulant ou d'un menu contextuel.

4.6.3.2 Fenêtre

Un menu de STYLE WINDOW se présente sous la forme d'une fenêtre non modale. Le LABEL du menu doit être utilisé comme légende pour la fenêtre.

4.6.3.3 Boîte de dialogue

Un MENU de STYLE DIALOG doit être une fenêtre modale. Si un dialogue contient une fenêtre subordonnée, l'application EDD doit gérer la fenêtre comme une boîte de dialogue secondaire modale. Le LABEL du menu doit être utilisé comme légende pour la fenêtre.

4.6.3.4 Tableau

Profils de communication: CS, FF, GPE, ISA100, PB, PN

> Un menu de STYLE TABLE doit être restitué comme un tableau. Chaque élément doit être représenté sous la forme d'une ou de plusieurs lignes, par exemple les ARRAYs. Les sous-menus forment une hiérarchie, qui doit être affichée.

Profils de communication: HART

> Un menu de STYLE TABLE doit être restitué comme un menu en cascade. Chaque élément doit être représenté sous la forme d'une ou de plusieurs lignes, par exemple les ARRAYs. Les sous-menus forment une hiérarchie, qui doit être affichée. Le libellé de l'élément MENU (y compris les VARIABLEs BIT_ENUMERATED) doit être affiché. Sélectionner l'élément MENU doit entraîner l'affichage des contenus.

4.6.3.5 Page

Un menu de STYLE PAGE à l'intérieur d'un élément WINDOW ou DIALOG doit être restitué comme un onglet et couvrir toute la largeur du canevas. Lorsque plusieurs pages (onglets) sont restituées dans la même fenêtre ou boîte de dialogue, chaque onglet doit pouvoir être sélectionné individuellement afin d'activer les contenus d'un onglet en particulier. Le LABEL du menu doit être utilisé comme légende pour l'onglet. Il doit y avoir un ROWBREAK implicite avant et après la PAGE.

L'application EDD doit interpréter les ROWBREAK ou d'autres éléments entre les pages comme des séparations horizontales.

Les COLUMNBREAKs et SEPARATORs entre les pages doivent être ignorés.

4.6.3.6 Groupe

Un menu de STYLE GROUP se présente sous la forme d'un cadre de groupe. Le LABEL du menu doit être utilisé comme légende pour le groupe. Afin de maintenir une disposition épurée de l'interface utilisateur, les règles restrictives ci-dessous s'appliquent.

- Le niveau d'imbrication des déclarations du GROUP est réduit au nombre de deux. En d'autres termes, un GROUP peut contenir d'autres GROUPs, mais ces derniers peuvent ne pas contenir davantage de GROUPs.
- Lorsqu'un GROUP de seconde couche contient un GROUP, ce GROUP doit être restitué soit comme un bouton, soit comme un lien hypertexte. Le LABEL doit apparaître sur le bouton ou en tant que lien hypertexte. Lorsque le bouton est activé ou que le lien hypertexte est ouvert, une boîte de dialogue correspondante doit s'ouvrir sur la fenêtre ou la boîte de dialogue en cours.

Au sein d'un GROUP, les méthodes et paramètres avec généralement un rapport logique sont regroupés. Le titre du GROUP indique cette relation, par exemple "AnalogInput 1". La même chaîne est également souvent utilisée dans les libellés des éléments du GROUP, par exemple "AnalogInput1_StaticRevision" ou "Analog Input1_Blockmode". Afin d'éviter cette duplication des informations, le développeur EDD peut réunir les éléments d'un GROUP dans une COLLECTION dans le but de remplacer les libellés d'origine. S'il doit être fait référence aux paramètres dans un GROUP, les paramètres peuvent être référencés avec la COLLECTION;

dans d'autres cas, en particulier sans contexte sémantique supplémentaire, les paramètres peuvent être référencés directement.

La Figure 8 représente un exemple d'EDD avec des COLLECTION MEMBERS au sein d'un MENU de STYLE GROUP.

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

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

MENU ai1_group_double                          /* Entraîne le double affichage de */
{
    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                          /* Entraîne l'affichage unique de */
{
    LABEL "Analog Input 1";
    STYLE GROUP;
    ITEMS
    {
        ai1_groupcol.strev,
        ai1_groupcol.blomo
    }
}
```

Figure 8 – Utilisation de COLLECTION MEMBERS dans les MENUs du STYLE GROUP

4.6.4 Eléments contenus

4.6.4.1 Vue d'ensemble

La manière dont les éléments contenus sont restitués dans un conteneur est décrite en 4.6.3.1. Tous les conteneurs restituent les éléments contenus de la même façon. Les éléments contenus, tels que les méthodes ou les variables, entre autres, ont un attribut LABEL. Afin de ne pas perturber la disposition de l'application EDD, par exemple avec des boutons surdimensionnés pour les méthodes, il convient que le développeur EDD n'utilise pas de très longues chaînes pour les LABELs. Pour cette raison, certaines applications EDD peuvent tronquer ces libellés. Une méthodologie doit toutefois exister, telle que "tool tip", "pop-up window", "expanding window", etc., afin d'afficher l'intégralité de la chaîne pour l'utilisateur.

4.6.4.2 Méthodes

Il convient de restituer une méthode comme un bouton ou un lien hypertexte. Il convient que le LABEL de la METHOD correspondante apparaisse sur le bouton ou en tant que lien hypertexte. Lorsque le bouton est activé ou que le lien hypertexte est ouvert, il convient que la METHOD correspondante s'exécute.

4.6.4.3 Variables

4.6.4.3.1 Généralités

Il convient que le libellé, la valeur et, si définies, les unités de la variable soient affichées de manière cohérente avec la définition de la VARIABLE correspondante.

Le traitement général des variables se présente comme suit.

- HANDLING = READ ou l'attribut de l'élément du menu est READ_ONLY: il convient que la valeur de la variable ne soit pas modifiable.
- HANDLING = WRITE: il convient de ne pas lire la valeur à partir de l'appareil en raison de son référencement comme élément contenu, par exemple dans un MENU.
- HANDLING = READ & WRITE: Il convient que la valeur soit modifiable.
- CLASS = DYNAMIC: il convient de mettre constamment à jour la valeur avec les valeurs actuelles relevées sur l'appareil. Si la valeur a été modifiée, aucune mise à jour continue ne doit plus se produire afin que la valeur modifiée ne soit pas remplacée par la valeur lue à partir de l'appareil.

Pour les VARIABLEs de type chaîne, il convient que la valeur ne soit pas tronquée ou répartie sur plusieurs lignes. Le retour à la ligne dans un mot est susceptible d'être confondu avec un véritable caractère "new line" ('\n') dans la valeur de chaîne.

Lorsqu'aucune valeur de VARIABLE n'est disponible à l'affichage, en raison par exemple d'une modification de VALIDITY ou de HANDLING, l'application EDD doit indiquer que la variable est dans un état non initialisé.

Le Tableau 20 montre la manière dont il convient que l'application EDD indique l'état non initialisé des VARIABLEs.

Tableau 20 – État non initialisé des VARIABLEs sur l'interface utilisateur

TYPE de VARIABLE	Indication d'état non initialisé
Valeurs numériques ENUMERATED Valeurs de chaînes Valeurs de date, d'heure et de durée	Vide avec l'indication d'un état "uncertain" proche du champ de valeur
BIT_ENUMERATED	Indication qu'aucun bit n'est défini avec indication de l'état "uncertain" proche du champ de valeur
BOOLEAN	Indication FALSE avec indication de l'état "uncertain" proche du champ de valeur

4.6.4.3.2 Variable de TYPE BIT_ENUMERATED

Les bits multiples peuvent être regroupés dans une variable énumérée par bit unique. Chaque bit peut avoir une signification distincte. Il est possible d'afficher chaque bit séparément et, de surcroît, d'afficher l'ensemble de la variable BIT_ENUMERATED. Les variables BIT_ENUMERATED peuvent être affichées sous forme de cases à cocher.

En cas d'affichage d'un bit de la variable BIT_ENUMERATED, il convient d'étendre la référence de variable avec un masque entre crochets et le LABEL de la variable n'est pas affiché.

Lorsque l'intégralité de la VARIABLE BIT_ENUMERATED s'affiche, les bits de la VARIABLE doivent s'afficher regroupés.

Lorsqu'une énumération de bits est de CLASS DIAGNOSTIC, il convient alors que les bits affichés soient traités comme un indicateur de statut. Il convient de présenter une indication rouge lorsque le bit est défini et verte lorsqu'il est redéfini.

Profils de communication: HART

> Lorsque l'intégralité de la VARIABLE BIT_ENUMERATED s'affiche, les bits de la VARIABLE doivent s'afficher à l'intérieur d'un cadre de groupe. Le LABEL de la VARIABLE BIT_ENUMERATED doit être utilisé comme libellé du cadre de groupe. Il n'est pas exigé que les applications EDD avec des contraintes d'affichage empêchant l'affichage de cadres de groupe satisfassent à cette exigence.

La Figure 9 représente un exemple d'EDD pour afficher les bits uniques d'une variable BIT_ENUMERATED.

```
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 – Affichage des bits uniques d'une variable BIT_ENUMERATED

La Figure 10 représente un exemple d'EDD indiquant les différences entre l'affichage complet d'une variable BIT_ENUMERATED et la façon dont elle peut s'afficher si tous les bits sont adressés de manière explicite.

```
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,           // il convient d'afficher tous les bits
                COLUMNBREAK,
                var1[0x1],      // Bit 0
                var1[0x2],      // Bit 1
                var1[0x4],      // Bit 2
                COLUMNBREAK,
                var1_group
        }
}
```

Figure 10 – Affichage des bits multiples d'une variable BIT_ENUMERATED

La Figure 11 montre de quelle manière peut apparaître le résultat de l'exemple d'EDD de la Figure 10 dans une application EDD.

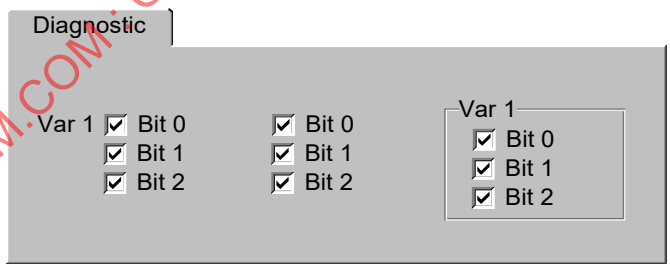


Figure 11 – Exemple d'application EDD pour une variable de type BIT_ENUMERATED

4.6.4.3.3 Variable de TYPE INDEX

Quand une VARIABLE de type INDEX est présentée à l'utilisateur, l'application EDD doit déterminer la valeur qui doit être affichée selon les règles suivantes:

1) Si la description dans les ELEMENTS référencés existe, cela doit être affiché autrement.
2) Si le LABEL dans les ELEMENTS référencés existe, cela doit être affiché autrement.
3) Le LABEL de l'ARRAY ou de la LIST doit être affiché avec la valeur numérique de l'INDEX, par exemple myarray[3].

Si la valeur INDEX n'est pas comprise dans la plage de la matrice correspondante, il convient qu'un texte informe l'utilisateur que la valeur n'est pas dans la plage.

Si la variable est modifiable, il convient de la présenter sous forme de boîte combinée.

4.6.4.3.4 Variable avec HANDLING WRITE

Conformément à l'attribut HANDLING, une VARIABLE avec HANDLING WRITE est une variable en "écriture seule". Cela signifie que la variable est modifiable et peut être écrite dans l'appareil, mais qu'il convient de ne pas la lire à partir de l'appareil.

Pour une session en ligne, l'application EDD doit afficher la valeur de VARIABLE comme étant non initialisée (voir Tableau 20), jusqu'à ce que la VARIABLE soit modifiée pendant la même session d'édition.

Pour une session hors ligne, l'application EDD doit afficher la valeur de VARIABLE à partir du cache actuel.

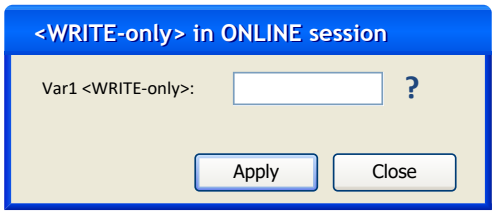
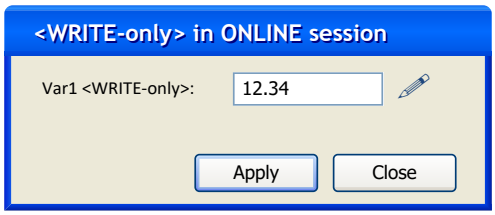
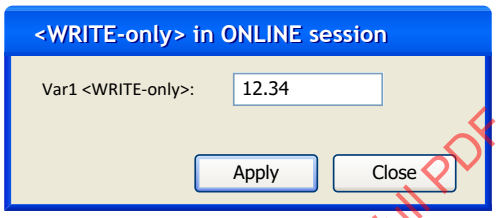
La Figure 12 représente un exemple d'EDD avec une variable en "écriture seule".

```
//--- UI description

MENU    device_root_menu
{
    LABEL       "Device setup";
    ITEMS
    {
        menu_WriteOnly_online
    }
}

MENU    menu_WriteOnly_online
{
    LABEL       "ONLINE session";
    STYLE       DIALOG;
    ITEMS
    {
        Var1
    }
}


//--- Data description

VARIABLE Var1
{
    LABEL       "Var1 <WRITE-only>";
    CLASS       DEVICE;
    TYPE        FLOAT
    {
        DEFAULT_VALUE   11.22;
    }
    HANDLING    WRITE;
}
```


Figure 12 – Exemple d'EDD avec une variable en "écriture seule" (HANDLING WRITE)

Le Tableau 21 représente un exemple d'étapes opérationnelles pour une variable en "écriture seule" dans une session en ligne.

Tableau 21 – Exemple de variable en "écriture seule" dans un dialogue en ligne

Etape	Interface utilisateur	Description
Etape 1: La boîte de dialogue est ouverte (état initial).		Comme "Var1" est une variable en "écriture seule", la valeur ne peut pas être lue à partir de l'appareil. "Var1" est affichée comme étant éditable et vide avec indication de l'état "uncertain" proche du champ de valeur.
Etape 2: Editer "Var" = 12.34		La nouvelle valeur de "Var1" est affichée avec indication de l'état "changed" proche du champ de valeur.
Etape 3: Appuyer sur "Apply" pour envoyer la valeur éditée		La nouvelle valeur de "Var1" est envoyée, c'est-à-dire transférée à l'appareil. La valeur envoyée s'affiche alors sans indication d'un état "uncertain" ou "changed".

4.6.4.4 Collections et enregistrements

Les COLLECTIONs et RECORDs doivent être affichés sous forme de MENU avec un STYLE GROUP. Les COLLECTIONs peuvent contenir des COLLECTIONs. Elles sont traitées comme des groupes imbriqués (voir 4.6.3.6). Les règles au sujet des groupes imbriqués s'appliquent également dans ce cas.

4.6.4.5 Matrices et listes

Profils de communication: CS, FF, GPE, ISA100, PB, PN

Les matrices de valeurs, les matrices de références et les LISTs doivent être restituées sous la forme d'une zone à faire défiler à l'intérieur du conteneur. Le LABEL de l'élément doit toujours être visible.

Profils de communication: FF

Il convient que chaque élément de la matrice soit affiché comme si chaque élément de la matrice était spécifiquement spécifié (dans l'ordre) comme des éléments de menu.

Profils de communication: HART

HART ne prend pas en charge les références de matrices et de listes comme des éléments contenus dans un MENU.

Il convient de traiter les éléments de LIST non existants référencés dans un MENU comme si leur valeur d'attribut VALIDITY était FALSE.

4.6.4.6 Affichages d'édition

Un EDIT_DISPLAY doit être rendu dans le conteneur sous forme de bouton ou de lien hypertexte. Le LABEL de l'EDIT_DISPLAY correspondant doit apparaître sur le bouton ou en tant que lien hypertexte.

Lorsque le bouton est activé ou que le lien hypertexte est ouvert, l'EDIT_DISPLAY correspondant doit s'ouvrir dans une fenêtre modale. Un EDIT_DISPLAY contient des DISPLAY_ITEMS et des EDIT_ITEMS. Les DISPLAY_ITEMS doivent être affichés en tant qu'éléments en lecture seule et les EDIT_ITEMS doivent être gérés comme des ITEMS depuis un MENU de STYLE DIALOG.

4.6.4.7 Graphiques

Il convient d'afficher un graphique de manière cohérente avec la définition du GRAPH correspondant. Se reporter aux règles de disposition définies en 4.7 et dans l'IEC 61804-3 pour obtenir des informations sur l'effet de l'attribut WIDTH sur la disposition du GRAPH. Il convient que l'application EDD mette à jour les données des WAVEFORMs lorsque l'utilisateur indique que la modification de la forme d'onde est achevée (bouton d'enregistrement, par exemple). Il convient que l'application EDD fournisse un mécanisme qui permet à l'utilisateur de restaurer la WAVEFORM d'origine sans mettre à jour les données (bouton d'annulation, par exemple).

4.6.4.8 Diagrammes

Il convient d'afficher un diagramme de manière cohérente avec la définition du CHART correspondant. Se reporter aux règles de disposition définies en 4.7 et dans l'IEC 61804-3 pour obtenir des informations sur l'effet de l'attribut WIDTH sur la disposition du CHART. Pour les CHARTs qui contiennent un axe temporel, il doit y avoir une indication de la durée du temps (par exemple secondes, minutes, temps absolu, etc.).

4.6.4.9 Images

Les applications portatives peuvent afficher une image référencée dans un MENU sous forme de lien hypertexte. Sinon, les applications EDD doivent restituer les images en utilisant les règles suivantes:

a) Il convient d'afficher les images référencées par le MENU.

b) Lorsqu'INLINE n'est pas spécifié, l'application EDD doit afficher une IMAGE dans sa taille d'origine. L'application EDD doit également insérer un ROWBREAK au-dessus de l'IMAGE, si des éléments existent avant l'IMAGE, et insérer un ROWBREAK sous l'IMAGE, si des éléments existent après l'IMAGE.

c) Lorsqu'INLINE est spécifié, l'application EDD doit restituer l'IMAGE en fonction du LAYOUT_TYPE (voir 4.7.2). Pour la "disposition de largeur égale des colonnes" (voir 4.7.2.2), l'application EDD doit réduire la taille d'une IMAGE si la largeur de l'IMAGE est supérieure à celle de la colonne où l'IMAGE se trouve. L'application EDD doit maintenir le facteur de forme d'origine des IMAGEs. Pour la "disposition de largeur optimale des colonnes" (voir 4.7.2.3), l'application EDD ne doit pas réduire la taille d'une IMAGE, qu'INLINE soit spécifié ou non (voir 4.7.5.10).

d) L'application EDD ne doit pas augmenter la taille d'une IMAGE.

e) Si l'IMAGE a l'attribut LINK défini, cela doit être indiqué à l'utilisateur.

f) L'application EDD doit afficher les IMAGES qui sont en transparence par rapport à l'arrière-plan du conteneur. Dans ce cas, la couleur du conteneur s'applique comme couleur d'arrière-plan de l'image.

Profils de communication: HART

La taille d'une image doit être inférieure à cent cinquante mille (150 000) octets et la taille de la somme de toutes les images d'une EDD ne doit pas dépasser cinq millions (5 000 000) d'octets.

Il convient que les images d'une largeur de 210 pixels ou moins puissent s'adapter à INLINE sans mise à l'échelle. Les applications EDD avec des contraintes d'affichage peuvent ne pas satisfaire à cette recommandation.

4.6.4.10 Texte statique

Il convient d'afficher le texte référencé par le MENU sous forme de texte non modifiable. Dans une vue en tableau, un long texte peut être tronqué ou réparti sur plusieurs lignes dans le menu. Pour toutes les autres vues, le texte est réparti sur plusieurs lignes s'il est plus long que la largeur de la boîte de dialogue, de la fenêtre, de la page, du groupe ou de la colonne.

4.6.4.11 Grille

Il convient d'afficher une grille de manière cohérente avec la définition de GRID correspondante. Le LABEL de la grille s'affiche en premier puis la zone de la grille en dessous. Se référer aux règles de disposition définies en 4.7 et dans l'IEC 61804-3 pour obtenir des informations sur l'effet de l'attribut WIDTH sur la disposition de la GRID. Il convient de prévoir des mécanismes pour faire défiler la zone de la grille si la largeur ou la hauteur est trop petite pour afficher l'ensemble des données.

4.6.4.12 Plugin

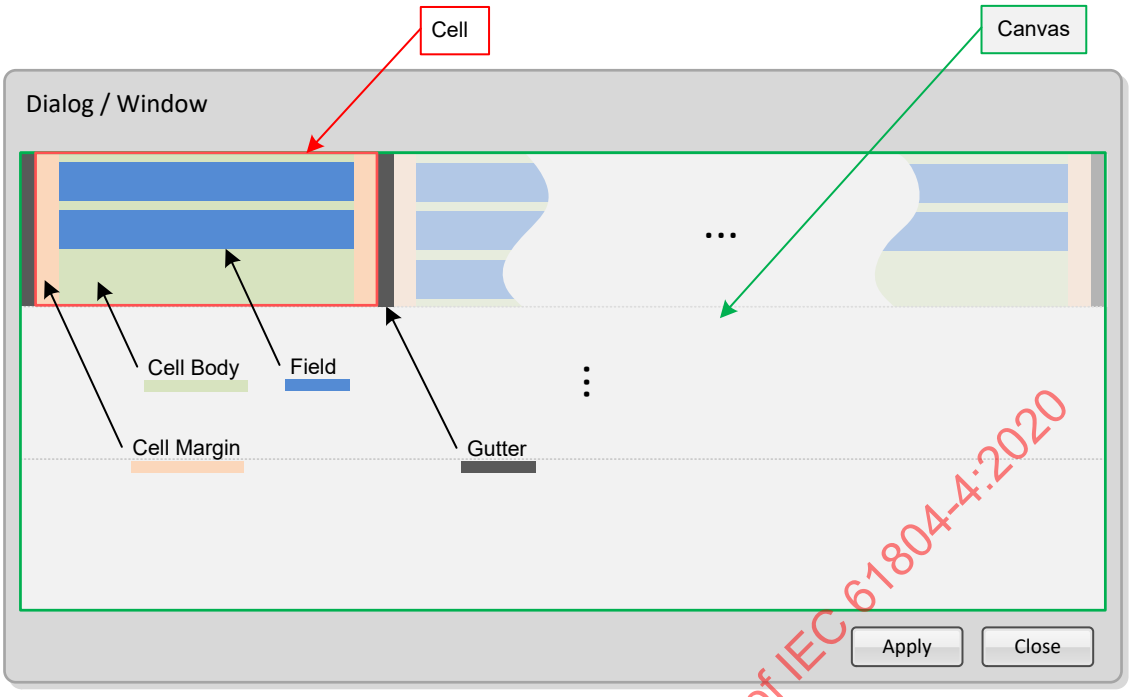
Il convient qu'un Plugin soit restitué comme un bouton ou un lien hypertexte. Il convient que le LABEL du PLUGIN correspondant apparaisse sur le bouton ou en tant que lien hypertexte. Lorsque le bouton est activé ou que le lien hypertexte est ouvert, il convient que le PLUGIN correspondant soit appelé.

4.7 Règles de disposition

4.7.1 Vue d'ensemble

Les règles de disposition sont des règles que l'application EDD doit appliquer pour l'agencement du contenu des fenêtres et boîtes de dialogue affichées à l'écran.

Les éléments de disposition de base sont représentés à la Figure 13 et décrits dans le Tableau 22.



Anglais	Français
Cell	Cellule
Canvas	Canevas
Dialog / Window	Boîte de dialogue / Fenêtre
Cell Body	Corps de cellule
Field	Champ
Cell Margin	Marge de cellule
Gutter	Blanc transversal

Figure 13 – Eléments de disposition de base

Tableau 22 – Description du contenu d'une disposition

Elément de disposition	Description et règles
Canevas	Zone de WINDOWs, DIALOGs ou PAGEs où l'application EDD doit arranger le contenu du MENU associé. Un canevas est organisé en lignes et en colonnes. La zone d'affichage par défaut du canevas doit être suffisamment large pour afficher au moins trois colonnes sans barre de défilement horizontale. Il convient que la zone d'affichage par défaut du canevas soit suffisamment grande pour afficher au moins 15 champs simples dans une seule colonne sans défilement vertical. Les applications EDD avec des contraintes d'affichage peuvent ne pas satisfaire à cette recommandation. Quand deux lignes ou plus ont le même nombre de colonnes, les sauts de colonne à l'intérieur de ces lignes doivent être alignés. L'organisation des colonnes dans le canevas doit être déterminée par les attributs de la construction LAYOUT_TYPE, COLUMNWIDTH_EQUAL ou COLUMNWIDTH_OPTIMIZED
Cellule	Zone contenant des champs entre deux sauts de ligne ou de colonne.
Champ	Zone occupée par un élément du menu.
Champ simple	Zone qui serait occupée dans une cellule par une VARIABLE numérique, son libellé, son unité, et une indication de statut (si affichée). Tous les champs simples d'une cellule doivent être affichés avec Tous les libellés visibles alignés Les valeurs alignées Toutes les unités visibles alignées Tous les statuts visibles alignés
Corps de cellule	Zone occupée par la collection de tous les champs d'une cellule. L'organisation des corps de cellule sur une ligne doit être déterminée par les attributs de la construction LAYOUT_TYPE, COLUMNWIDTH_EQUAL ou COLUMNWIDTH_OPTIMIZED
Marge de cellule	Espace vide entre la limite et le corps de la cellule Toutes les marges de cellule sur une ligne doivent être égales
Blanc transversal	Espace vide entre deux cellules et entre une cellule et le bord du canevas d'une ligne donnée Tous les blancs transversaux d'une ligne doivent être égaux

4.7.2 Contrôle de la disposition par l'attribut LAYOUT_TYPE

4.7.2.1 Vue d'ensemble

L'attribut LAYOUT_TYPE fait partie des informations d'identification d'EDD et spécifie la manière dont l'application EDD doit restituer toutes les interfaces utilisateurs graphiques définies dans l'EDD.

4.7.2.2 Disposition de largeur égale des colonnes (COLUMNWIDTH_EQUAL)

Lorsque le LAYOUT_TYPE est défini sur COLUMNWIDTH_EQUAL, chaque MENU dont le STYLE est évalué comme étant WINDOW, DIALOG, PAGE ou GROUP doit être restitué en utilisant des colonnes de largeur égale.

La Figure 14 représente un exemple de disposition avec des colonnes de largeur égale sur chaque ligne.

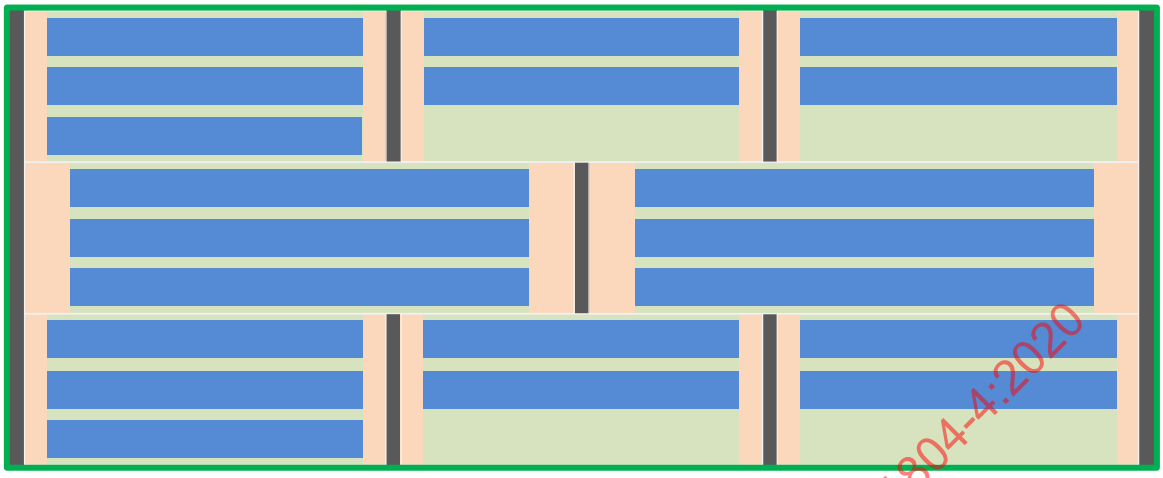


Figure 14 – Exemple de disposition avec des colonnes de largeur égale

Sur toute ligne, la largeur de chaque colonne doit être égale.

4.7.2.3 Disposition de largeur optimale des colonnes (COLUMNWIDTH_OPTIMIZED)

Lorsque le LAYOUT_TYPE est défini sur COLUMNWIDTH_OPTIMIZED, chaque MENU dont le STYLE est évalué comme étant WINDOW, DIALOG, PAGE ou GROUP doit être restitué en utilisant des colonnes de largeur optimale.

La Figure 15 représente un exemple de disposition avec des colonnes de largeur optimale par rapport au contenu.

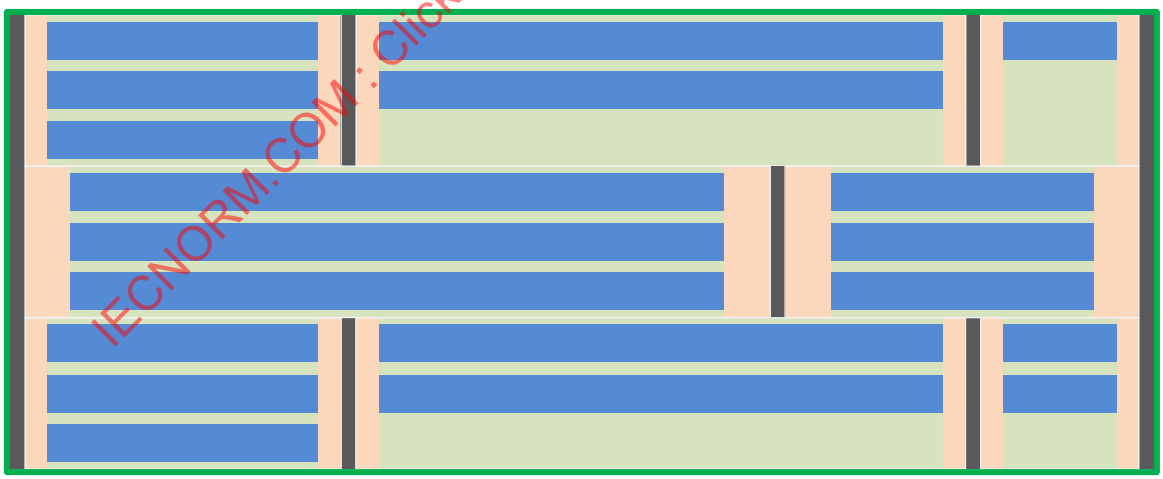


Figure 15 – Exemple de disposition avec des colonnes de largeur optimale

L'application EDD doit arranger la disposition en fonction du nombre réel de COLUMNBREAKs et de la largeur des conteneurs. La largeur de chaque colonne doit être optimisée en fonction de l'élément de menu le plus large dans la colonne. Toutes les lignes ayant le même nombre de colonnes doivent être prises en compte dans ce calcul.

Contrairement à la disposition de largeur égale des colonnes, les éléments de menu ne sont pas déplacés automatiquement vers la ligne suivante lorsqu'un élément de menu couvrirait une colonne supérieure à la 5e. Pour commencer une nouvelle ligne dans une disposition de largeur optimale des colonnes, un ROWBREAK est exigé.

La Figure 16 représente une cellule de disposition qui contient une chaîne constante (a), une VARIABLE en lecture seule (b), une METHOD (c) restituée sous forme de bouton et une VARIABLE éditable (d). Les libellés des VARIABLEs sont affichés à gauche. La cellule divisée en zones distinctes pour le libellé, le champ d'entrée qui contient la valeur, le statut de la variable et l'unité.

Les chaînes constantes, les éléments de menu restitués sous forme de bouton et les valeurs de la VARIABLE doivent être alignés et affichés dans la zone du champ d'entrée.

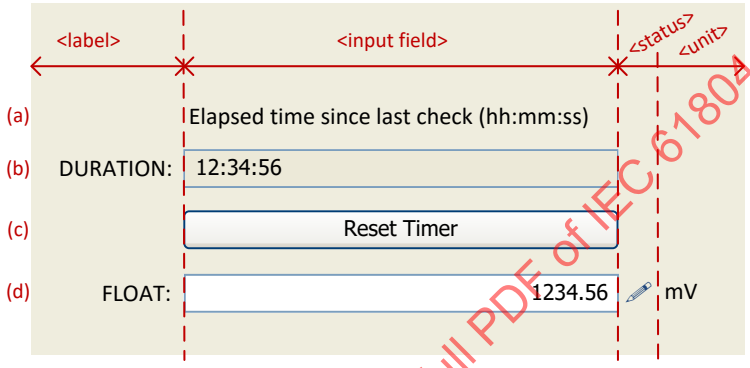


Figure 16 – Corps de cellule dans une disposition avec des colonnes de largeur optimale (libellé à gauche)

La Figure 17 représente une cellule de disposition qui contient les mêmes éléments de menu que celle de la Figure 16, mais où les libellés des VARIABLEs sont affichés en haut. La zone du libellé et la zone du champ d'entrée sont fusionnées.

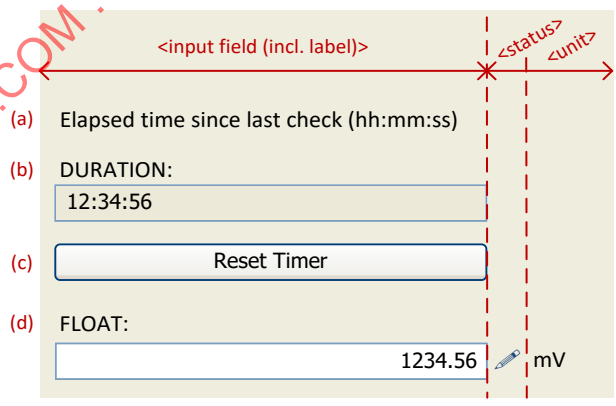


Figure 17 – Corps de cellule dans une disposition avec des colonnes de largeur optimale (libellé en haut)

Le Tableau 23 récapitule les limites de largeur minimale et maximale pour les champs d'entrée en fonction du type d'élément de menu.

Tableau 23 – Largeur minimale et maximale des champs d'entrée couvrant une colonne

Elément de disposition	Largeur minimale / maximale pour les champs d'entrée (valeurs approx. en pixels)
Eléments de menu restitués sous forme de bouton	Min. 80 pixels / pas de limite max.
Chaînes constantes	Pas de limite min. / max. 350 pixels
VARIABLEs	Min. 100 pixels (peut être plus large en fonction de la commande de l'interface utilisateur) / max. 350 pixels

La largeur optimale de colonne doit être calculée de la manière suivante:

1) La largeur exigée de chaque élément de menu doit être déterminée en fonction des règles suivantes:
   - Pour les éléments de menu restitués sous forme de bouton ou de lien hypertexte, par exemple des METHODs, la largeur doit être déterminée par le libellé.
   - Pour les chaînes constantes, la largeur doit être déterminée par la longueur réelle de la chaîne. Pour les très longues chaînes dépassant la largeur max. d'un champ d'entrée, le retour à la ligne automatique dans un mot doit être pris en charge.
   - Pour les VARIABLEs, la largeur doit être calculée en additionnant la largeur demandée pour le libellé, pour la valeur et l'unité et, si affiché, pour le statut de la variable. La largeur du champ d'entrée qui affiche la valeur dépend du type de données de la variable. Pour les variables ENUMERATED ou BIT_ENUMERATED, la largeur doit être déterminée par l'élément d'énumération le plus large.
     Puisque dans une disposition qui présente le libellé en haut, les zones du libellé et du champ d'entrée sont fusionnées, il convient de déterminer la zone du champ d'entrée en tenant compte de la largeur du libellé et de la largeur du champ d'entrée. Lorsque le libellé est plus large que la largeur demandée du champ d'entrée, il convient d'étendre le champ d'entrée en fonction de la largeur du libellé.
     Il convient de ne pas tronquer le libellé et l'unité.
   - Pour les IMAGEs, la largeur doit être déterminée par la largeur d'origine de l'image et ne doit pas être ajustée, que l'image référencée soit marquée par le qualificatif INLINE ou non. Si l'IMAGE exige une largeur supérieure à la largeur disponible, la colonne doit être étendue.
   - Pour les éléments restitués sous forme de GROUP, la largeur doit être déterminée en fonction du contenu et en tenant compte du libellé du menu et des marges dans le cadre de groupe. Si le libellé du menu est plus large que le contenu, la largeur du cadre de groupe doit être déterminée en fonction du libellé du menu.
2) Par ailleurs, les limites de largeur minimale et maximale du champ d'entrée récapitulées dans le Tableau 23 doivent être prises en compte. Noter que, dans les dispositions présentant le libellé en haut, le champ d'entrée peut être plus large que la limite de largeur maximale.
3) La largeur réelle de la colonne doit être évaluée comme étant la largeur de l'élément le plus large de la colonne.
4) Pour une disposition bien arrangée, les éléments de chaque colonne doivent être étendus de manière à couvrir toute la colonne en étendant les champs d'entrée respectifs (voir Figure 16).
   Pour les éléments de menu non restitués avec un champ d'entrée, toute la zone de l'élément de menu doit être étendue.
5) Dans une disposition à plusieurs lignes et plusieurs colonnes, le nombre de colonnes peut varier d'une ligne à l'autre. Le contenu des lignes ayant le moindre nombre de colonnes doit être étendu de façon à ce que chaque ligne couvre la même largeur. Dans ce cas, les champs d'entrée peuvent être plus larges que la largeur maximale respective spécifiée dans le Tableau 23.

6) Après avoir calculé la largeur de chaque colonne, la largeur du canevas qui en résulte doit être calculée en additionnant la largeur de chaque colonne. Lorsque la largeur du canevas qui en résulte est supérieure à la largeur de l'écran, une barre de défilement horizontale doit être prévue.

Lorsque l'utilisateur modifie la taille du canevas en modifiant les dimensions de la fenêtre ou de la boîte de dialogue, les règles suivantes doivent s'appliquer:

- En cas de réduction, la largeur de colonne optimale calculée doit s'appliquer en tant que largeur minimale de chaque colonne.
- En cas d'extension, le gain de largeur doit être réparti de manière régulière parmi l'ensemble des colonnes.
- Le contenu de chaque colonne doit toujours couvrir toute la colonne en ajustant la largeur du champ d'entrée respectif.

Lorsque la largeur exigée varie en raison d'une expression conditionnelle, la disposition qui en résulte peut s'étendre, mais pas se réduire. Si la disposition qui en résulte exige une largeur inférieure, la largeur de dépassement doit être répartie de manière régulière parmi l'ensemble des colonnes.

4.7.3 Règles de disposition pour les attributs WIDTH et HEIGHT

4.7.3.1 Généralités

VARIABLE, CHART, GRAPH et GRID possèdent les attributs WIDTH et HEIGHT. L'attribut WIDTH définit le nombre de colonnes à afficher. L'attribut HEIGHT définit la taille verticale de l'affichage en tant que multiple de la hauteur minimale d'un champ simple (voir Tableau 24). La valeur par défaut de WIDTH et de HEIGHT est MEDIUM pour CHART, GRAPH, GRID; elle est XXX_SMALL pour la largeur et la hauteur de VARIABLE.

Disposition de largeur égale des colonnes:

> Il convient de diviser la largeur de l'application EDD jusqu'à 5 colonnes en fonction du nombre de COLUMNBREAK et de la largeur des conteneurs. Les colonnes qui atteindraient une colonne supérieure à la 5^{e} colonne doivent être déplacées vers la ligne suivante par un ROWBREAK implicite.
>
> Un ROWBREAK implicite a le même comportement qu'un ROWBREAK explicite défini.

Disposition de largeur optimale des colonnes:

> Il convient de diviser le canevas en fonction du nombre de COLUMNBREAKs et de la largeur des conteneurs. Les colonnes peuvent atteindre une colonne supérieure à la 5^{e} sans être déplacées vers la ligne suivante par un ROWBREAK implicite.

Les attributs WIDTH et HEIGHT définissent le nombre de colonnes que l'élément peut comprendre et la hauteur en multiples de la hauteur minimale possible pour un champ simple, par exemple une variable, une chaîne constante, un menu et une référence de méthode (voir Tableau 24).

Tableau 24 – Etendue et applicabilité de WIDTH et de HEIGHT

Valeurs WIDTH et HEIGHT	WIDTH	HEIGHT	Applicabilité
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

4.7.3.2 VARIABLEs dont l'attribut WIDTH couvre des colonnes (disposition de largeur optimale des colonnes)

L'attribut WIDTH d'un élément spécifie le nombre de colonnes couvertes par l'élément. Par conséquent, les règles suivantes doivent s'appliquer:

1) S'il s'étend sur des colonnes normalement larges, l'élément de menu doit être étendu de manière à couvrir l'ensemble des colonnes correspondantes (voir Figure 44).
2) Lorsqu'il couvre des colonnes vides ou très étroites, l'incrément de largeur de cet élément de menu doit être divisé de manière régulière sur l'ensemble des colonnes correspondantes. L'application EDD doit appliquer les incréments de largeur en tenant compte de la largeur minimale des champs d'entrée (voir Tableau 23).
3) Les colonnes vides ou étroites doivent être étendues en conséquence (voir Figure 19).
4) Pour les VARIABLEs qui couvrent plusieurs colonnes, seul le champ d'entrée doit être étendu.

La Figure 19 représente une disposition qui contient 2 lignes, la 1re avec 5 colonnes étroites et la 2e avec des variables couvrant un nombre différent de colonnes en définissant l'attribut WIDTH. La Figure 18 représente le code source correspondant.

```
MENU menu_Layout_Width_on_Variables
{
       LABEL  "WIDTH on Variables";
       STYLE DIALOG;
       ITEMS
       {
              "1",   COLUMNBREAK,          // 5 colonnes "étroites"
              "2",   COLUMNBREAK,
              "3",   COLUMNBREAK,
              "4",   COLUMNBREAK,
              "5",
              ROWBREAK,
              var_ASCII,                   // Variable ASCII sans WIDTH → par défaut
              var_ASCII_Width_X_SMALL,     // Variable ASCII avec 'WIDTH X_SMALL'
              var_ASCII_Width_SMALL,            // Variable ASCII avec 'WIDTH SMALL'
              var_ASCII_Width_MEDIUM,      // Variable ASCII avec 'WIDTH MEDIUM'
              var_ASCII_Width_LARGE,            // Variable ASCII avec 'WIDTH LARGE'
              var_ASCII_Width_X_LARGE      // Variable ASCII avec 'WIDTH X_LARGE'
       }
}

//--- Définition de VARIABLE exemplaire
VARIABLE var_ASCII_Width_X_SMALL
{
       LABEL  "ASCII (X_SMALL = 1C)";
       CLASS  LOCAL;
       TYPE   ASCII(20);
}
```

Figure 18 – Code source EDD pour une disposition avec des VARIABLEs couvrant des colonnes

La largeur de la 1re colonne est déterminée en additionnant la largeur:

- du libellé le plus large, qui est "ASCII (MEDIUM = 3C)";
- de la valeur de la 1e variable "ASCII (Default = 1C)";
- d'aucune unité à prendre en compte.

La largeur des colonnes 2 à 5 est déterminée par les incréments de largeur en raison de la couverture de colonnes vides ou étroites.

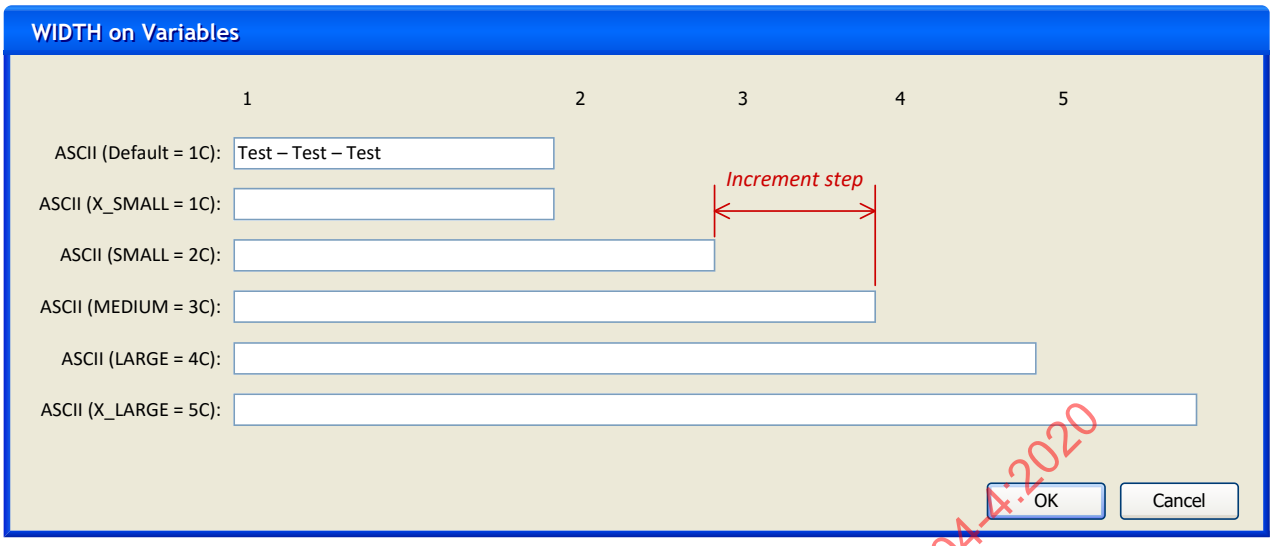


Figure 19 – Disposition avec VARIABLEs couvrant plusieurs colonnes

4.7.3.3 CHART et GRAPH dont l'attribut WIDTH couvre des colonnes (disposition de largeur optimale des colonnes)

Selon la déclaration du CHART ou du GRAPH dans l'EDD, il convient que l'application EDD définisse une largeur minimale par rapport à la largeur pour structurer les éléments (par exemple ordonnées, légendes) et les caractéristiques de la commande de l'interface utilisateur utilisée pour représenter le CHART ou le GRAPH.

Pour réduire le plus possible le besoin d'une barre de défilement horizontale, il convient que l'application EDD définisse une largeur maximale pour un CHART ou un GRAPH qui couvre 5 colonnes en tenant compte des caractéristiques réelles de l'écran.

L'incrément qui en résulte comme gain de largeur minimal par colonne est: (Largeur max. – largeur min.) / 4.

L'attribut WIDTH d'un élément spécifie le nombre de colonnes couvertes par le CHART ou le GRAPH. Par conséquent, les règles suivantes doivent s'appliquer:

1) Lorsqu'il couvre des colonnes vides ou très étroites, le gain de largeur du CHART ou du GRAPH doit être divisé de manière régulière sur l'ensemble des colonnes correspondantes. L'application EDD doit appliquer plusieurs incréments de largeur conformément à l'attribut WIDTH.
2) S'il s'étend sur des colonnes en consommant une largeur supérieure à un incrément, le CHART ou le GRAPH doit être étendu pour couvrir l'ensemble des colonnes correspondantes.
3) La largeur maximale peut être dépassée afin que toutes les lignes de la disposition aient la même largeur.

4.7.3.4 GRID dont l'attribut WIDTH couvre des colonnes (disposition de largeur optimale des colonnes)

Pour réduire le plus possible le besoin d'une barre de défilement horizontale, il convient que l'application EDD définisse une largeur maximale pour une GRID qui couvre 5 colonnes en tenant compte des caractéristiques réelles de l'écran et de l'incrément correspondant comme gain de largeur minimal par colonne.

L'attribut WIDTH d'un élément spécifie le nombre de colonnes couvertes par la GRID. Par conséquent, les règles suivantes doivent s'appliquer:

1) Lorsqu'il couvre des colonnes vides ou très étroites, le gain de largeur de la GRID doit être divisé de manière régulière sur l'ensemble des colonnes correspondantes. L'application EDD doit appliquer plusieurs incréments de largeur conformément à l'attribut WIDTH.
2) Si elle s'étend sur des colonnes en consommant une largeur supérieure à un incrément, la GRID doit être étendue pour couvrir l'ensemble des colonnes correspondantes.
3) L'application EDD doit tenir compte de la largeur réellement exigée pour la GRID selon le contenu (VECTORS) comme la largeur maximale de la GRID. Cette largeur peut être dépassée afin que toutes les lignes de la disposition aient la même largeur.

4.7.4 Règles de disposition pour les attributs COLUMNBREAK et ROWBREAK

4.7.4.1 Disposition pour les éléments qui dépassent

L'application EDD doit insérer l'ensemble des éléments du menu dans une seule colonne, les uns à la suite des autres, jusqu'à rencontrer un COLUMNBREAK. Le COLUMNBREAK entraîne l'insertion du prochain élément du menu en haut de la colonne suivante de la même ligne. Si l'un des éléments du menu occupe une zone dans la prochaine colonne, alors l'élément du menu est inséré dans la ligne immédiatement inférieur. Voir la Figure 20 pour un exemple de code source EDD et la Figure 21 pour la disposition correspondante.

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

Figure 20 – Exemple de code source EDD pour la disposition des éléments qui dépassent

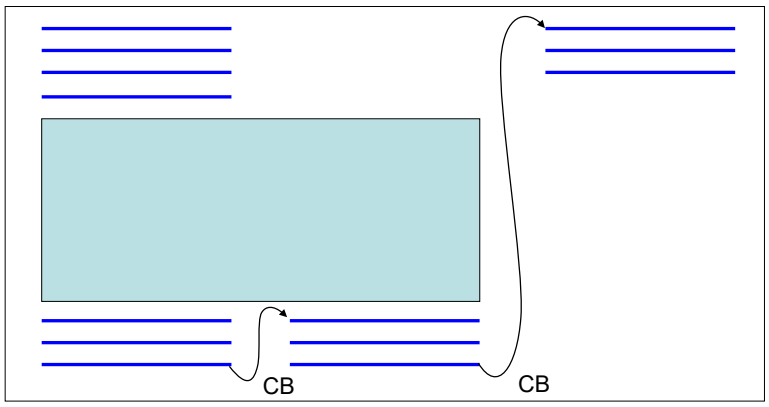


Figure 21 – Disposition pour les éléments qui dépassent

4.7.4.2 Disposition pour les lignes partiellement remplies

L'application EDD doit insérer l'ensemble des éléments du menu dans une seule colonne, les uns à la suite des autres, jusqu'à rencontrer un ROWBREAK. Un ROWBREAK entraîne l'insertion du prochain élément du menu dans la ligne suivante de la 1re colonne (de la même manière qu'un retour chariot). Le ROWBREAK doit également mettre en place une barrière horizontale afin que tout futur COLUMNBREAK empêche l'insertion d'un élément du menu au-dessus de cette ligne; voir la Figure 22 pour un exemple de code source EDD et la Figure 23 pour la disposition correspondante.

```
MENU menu1
{
        LABEL "menu 1";
        STYLE WINDOW;
        ITEMS
        {
                Element1,
                Element2,
                Element3,
                COLUMNBREAK,
                Element4,
                Element5,
                Element6,
                Element7,
                ROWBREAK,
                SmallGraph8,   //WIDTH SMALL, HEIGHT SMALL, occupe 2 colonnes
                Element9,
                Element10,
                Element11,
                COLUMNBREAK,
                Element12,
                Element13,
                Element14,
                COLUMNBREAK,
                Element15,
                Element16,
                Element17,
        }
}
```

Figure 22 – Exemple de code source EDD pour la disposition des lignes partiellement remplies

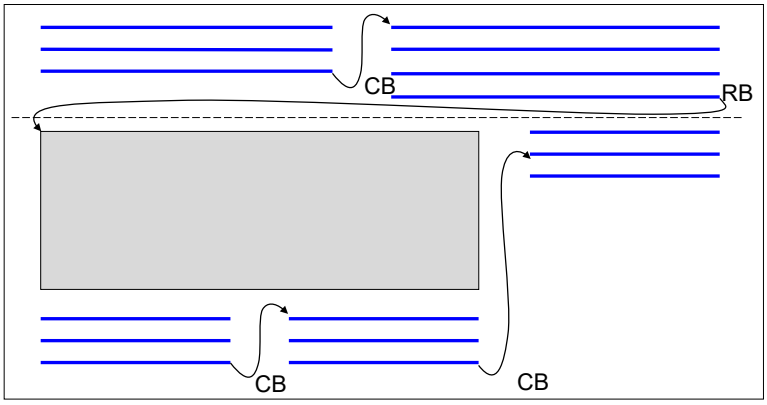


Figure 23 – Disposition pour les lignes partiellement remplies

La Figure 25 montre que les éléments de menu dans les lignes avec deux colonnes (lignes 1 et 3) ont étendu l'espace de cellule disponible de telle sorte que la ligne couvre la même largeur que les lignes avec trois colonnes. La Figure 24 représente l'exemple du code source EDD correspondant.

```
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 24 – Exemple de code source EDD pour la disposition des lignes partiellement remplies

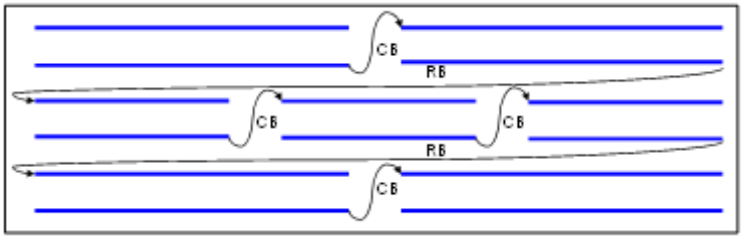


Figure 25 – Disposition pour les lignes partiellement remplies

4.7.4.3 Disposition pour les éléments surdimensionnés

L'application EDD doit insérer l'ensemble des éléments du menu dans une seule colonne, les uns à la suite des autres, jusqu'à rencontrer un COLUMNBREAK. Le COLUMNBREAK entraîne l'insertion du prochain élément du menu dans la plus haute ligne de la colonne suivante. Si l'élément du menu à insérer est plus large que le nombre de colonnes disponibles, celui-ci est alors inséré dans la ligne la plus basse de la 1re colonne (ROWBREAK implicite).

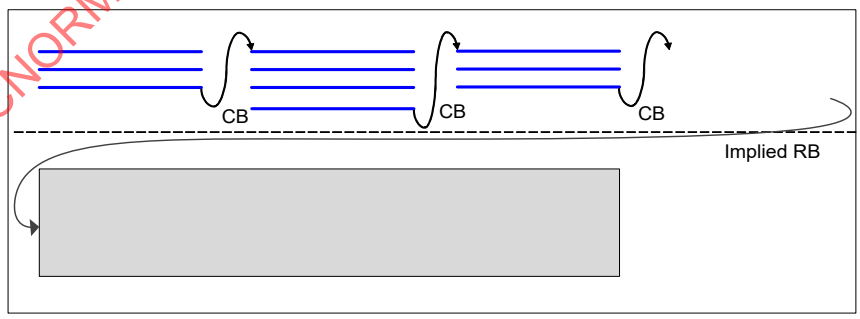
Avec l'exemple suivant, il convient d'expliquer la disposition des éléments surdimensionnés pour les types de dispositions fondés sur l'exemple de code source EDD donné à la Figure 26.

```
MENU disposition_for_oversized_elements
{
        LABEL "disposition for oversized elements";
        STYLE WINDOW;
        ITEMS
        {
                Element1,
                Element2,
                Element3,
                COLUMNBREAK,
                Element4,
                Element5,
                Element6,
                COLUMNBREAK,
                Element7,
                Element8,
                Element9,
                COLUMNBREAK,    //Entraîne un ROWBREAK implicite
                MediumGraph10   //WIDTH MEDIUM, HEIGHT MEDIUM, occupe 3 colonnes
        }
}
```

Figure 26 – Exemple de code source EDD pour la disposition des éléments surdimensionnés

Disposition de largeur égale des colonnes:

En raison du maximum de 5 colonnes pour la disposition de largeur égale des colonnes, l'élément surdimensionné est déplacé dans la 1re colonne de la ligne suivante (voir Figure 27).



Anglais	Français
Implied RB	RB implicite

Figure 27 – Elément surdimensionné dans une disposition avec des colonnes de largeur égale

Disposition de largeur optimale des colonnes:

Etant donné que le nombre maximal de 5 colonnes ne s'applique pas à la disposition de largeur optimale des colonnes, l'élément surdimensionné n'est pas déplacé vers la ligne suivante, mais vers la colonne suivante de la même ligne afin de couvrir les colonnes 4 à 6 (voir Figure 28).

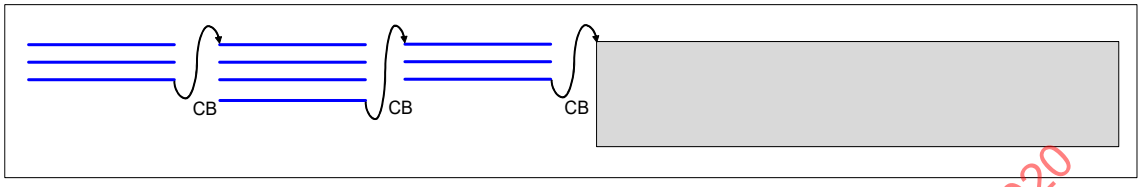


Figure 28 – Elément surdimensionné dans une disposition avec des colonnes de largeur optimale

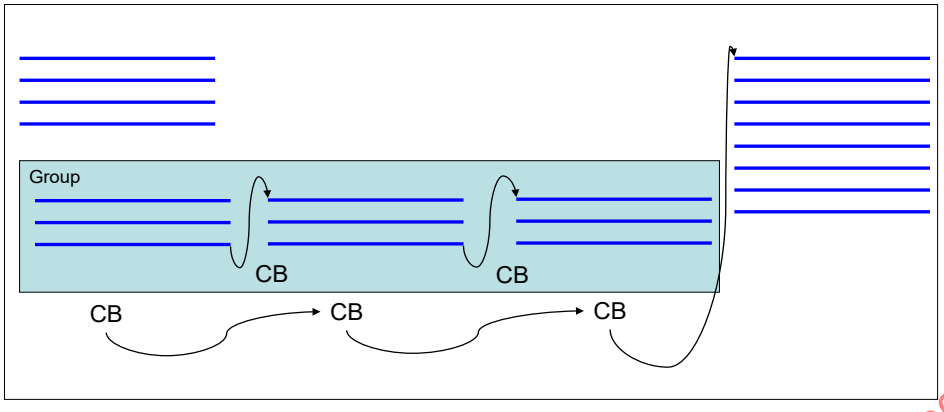
Pour déplacer l'élément surdimensionné dans la 1re colonne de la ligne suivante, comme le montre la Figure 27, le développeur EDD doit utiliser un ROWBREAK plutôt qu'un COLUMNBREAK pour contrôler la position de l'élément surdimensionné.

4.7.4.4 Disposition pour les colonnes en groupe superposé

L'application EDD doit insérer l'ensemble des éléments du menu dans une seule colonne, les uns à la suite des autres, jusqu'à rencontrer un COLUMNBREAK. Le COLUMNBREAK entraîne l'insertion du prochain élément du menu dans la plus haute ligne de la colonne suivante. Si l'élément de menu se trouve dans un menu imbriqué (comme un GROUP), l'élément reste alors dans le menu parent; voir Figure 29 pour un exemple de code source EDD et Figure 30 pour la disposition correspondante.

```
MENU disposition_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 29 – Exemple de code source EDD pour une disposition des colonnes en groupe superposé



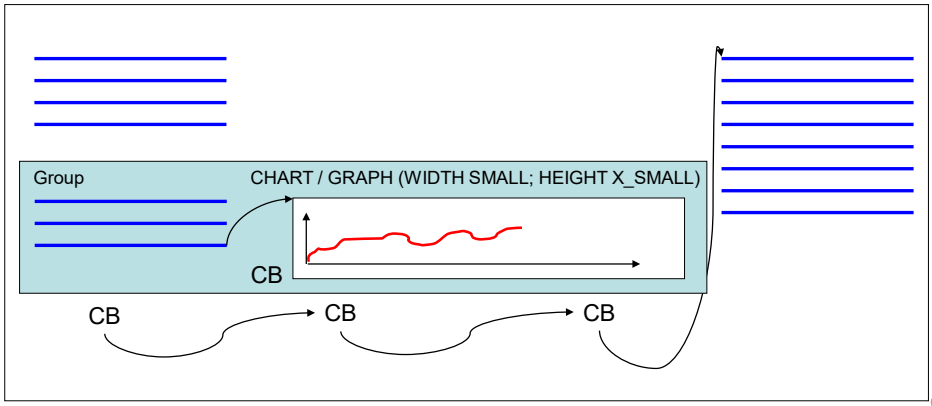
Anglais	Français
Group	Groupe

Figure 30 – Disposition pour les colonnes en groupe superposé

Les graphiques à l'intérieur de groupes doivent être traités de manière équivalente à plusieurs colonnes à l'intérieur d'un groupe imbriqué; voir la Figure 31 pour un exemple de code source EDD et la Figure 32 pour la disposition correspondante.

```
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,
                GroupWithThreeElementsOneColumnsBreakChartWidthSmall,
                COLUMNBREAK,
                COLUMNBREAK,
                COLUMNBREAK,
                Element8,
                Element9,
                Element10,
                Element11,
                Element12,
                Element13,
                Element14,
                Element15
        }
}
```

Figure 31 – Exemple de code source EDD pour la disposition des colonnes avec GRAPHs en groupe superposé



Anglais	Français
Group	Groupe

Figure 32 – Disposition pour les colonnes avec GRAPHs en groupe superposé

4.7.5 Exemples de disposition

4.7.5.1 Généralités

Les exemples donnés en 4.7.5 représentent des dispositions restituées selon l'exemple de code source. Des exemples de dispositions représentant les différences spécifiques entre les dispositions de largeur optimale des colonnes et les dispositions de largeur égale des colonnes sont donnés dans les paragraphes indiquant "disposition de largeur optimale des colonnes".

4.7.5.2 Disposition pour une seule colonne

La plus simple des dispositions consiste en une liste de variables, méthodes, affichages d'édition, textes statiques, graphiques et diagrammes.

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

Figure 33 – Exemple d'EDD pour un menu de présentation

L'EDD représentée à la Figure 33 affiche les éléments verticalement dans une fenêtre. Les éléments sont affichés en commençant par le côté gauche de l'écran et utilisent tout l'espace qui leur est nécessaire dans la fenêtre (voir Figure 34).

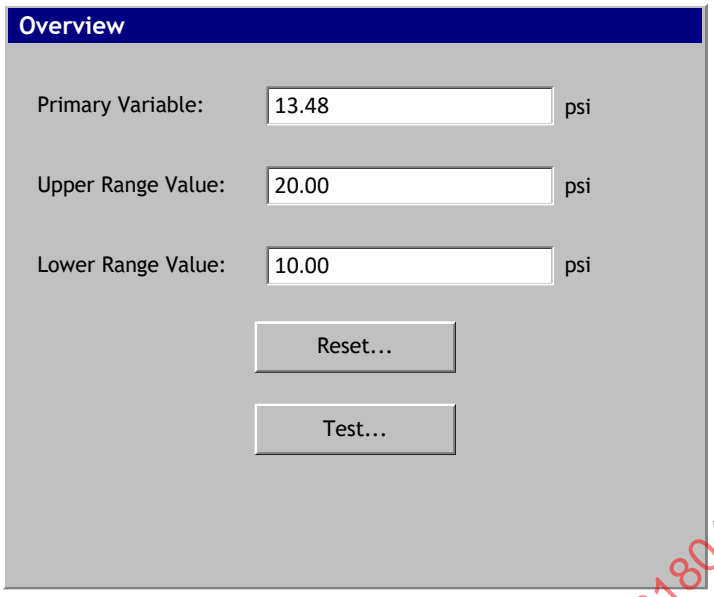


Figure 34 – Exemple d'application EDD pour une fenêtre de présentation

4.7.5.3 Disposition pour une seule colonne (disposition de largeur optimale des colonnes)

Dans une disposition pour une seule colonne, tous les éléments de menu sont arrangés dans une colonne. La Figure 36 représente une disposition pour une seule colonne qui contient des éléments de menu avec des exigences de largeur différentes. La Figure 35 représente le code source correspondant.

```
MENU DIALOG_LayoutTest_SingleColumn
{
       LABEL          "Single Column Layout";
       STYLE          DIALOG;
       ITEMS
       {
              "Elapsed time since last check",
              var_DURATION,
              method_ResetTimer,
              var_ASCII,
              var_FLOAT,
              var_ENUM_Color
       }
}
```

Figure 35 – Code source EDD pour une disposition avec éléments de menu couvrant une seule colonne

La largeur de toute la colonne est déterminée en additionnant la largeur:

- du libellé le plus large, qui est "ENUMERATED";
- de l'élément de menu le plus large, qui est "var_ENUM_Color" et qui contient un élément d'énumération extra-large;
- de l'unité la plus large, qui est "hh:mm:ss";
- du statut de la variable, le cas échéant.

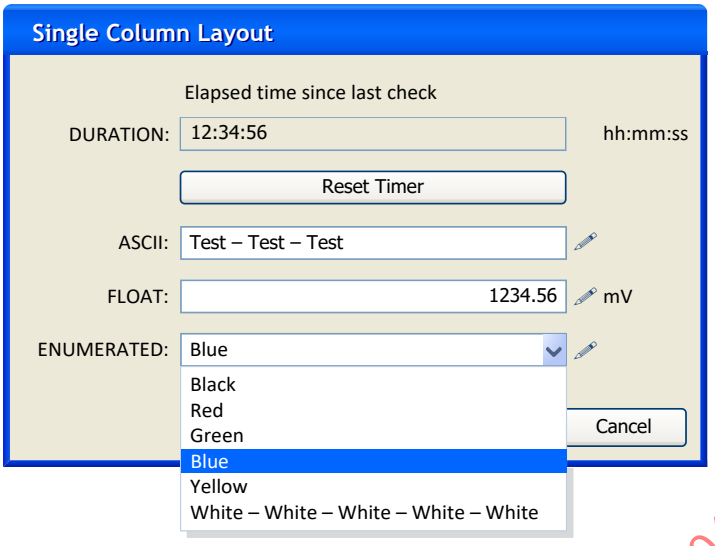


Figure 36 – Exemple de disposition avec éléments de menu couvrant une seule colonne

4.7.5.4 Disposition pour plusieurs colonnes et plusieurs lignes

Plusieurs colonnes de variables, méthodes, groupes, images, graphiques et diagrammes peuvent également être utilisées (voir Figure 37).

```
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,                /* saut de colonne */
        tag,                        /* variable */
        units,                      /* variable */
        damping,                    /* variable */
        transfer_function           /* variable */
    }
}
```

Figure 37 – Exemple d'EDD qui utilise un COLUMNBREAK

Un COLUMNBREAK dans la Figure 37 est utilisé pour indiquer un saut de colonne. Tous les éléments qui précèdent le COLUMNBREAK sont placés dans une même colonne; tous les éléments suivant le COLUMNBREAK sont placés dans la colonne suivante (voir Figure 38).

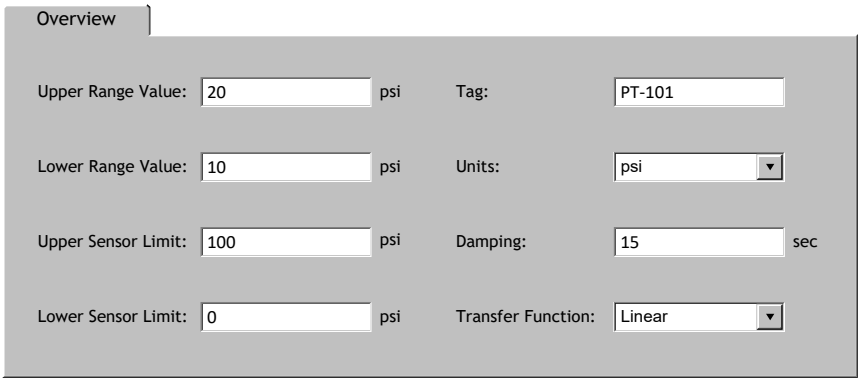


Figure 38 – Exemple d'application EDD pour une fenêtre de présentation

Un ROWBREAK dans la Figure 39 est utilisé pour placer les éléments de menu suivants en commençant par le côté gauche (voir Figure 40).

```
MENU overview_page
{
    LABEL "Overview";
    STYLE PAGE;
    ITEMS
    {
        tag,                          /* variable */
        ROWBREAK,
        range_values,                 /* groupe */
        COLUMNBREAK,
        sensor_limits,                /* groupe */
        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 39 – Exemple d'EDD pour une fenêtre de présentation

La Figure 40 représente le résultat de la Figure 39 dans une application EDD. Les encadrés en pointillés bleus indiquent l'espace disponible.

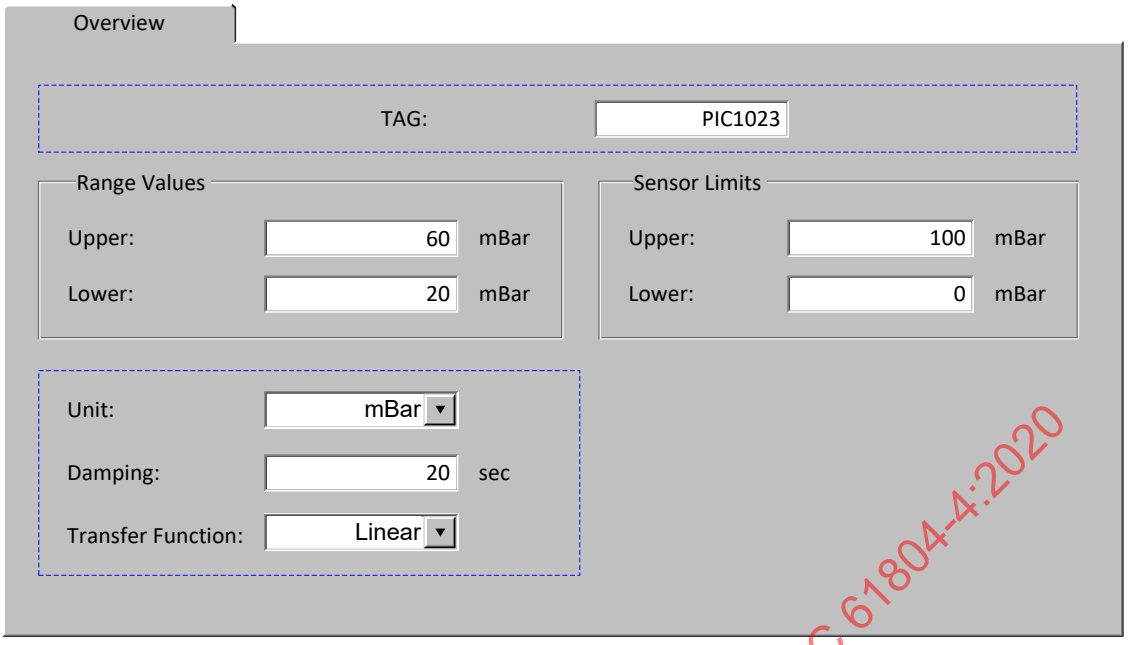


Figure 40 – Exemple d'application EDD pour une fenêtre de présentation

4.7.5.5 Disposition pour plusieurs colonnes et plusieurs lignes (disposition de largeur optimale des colonnes)

Dans une disposition pour plusieurs colonnes, les éléments de menu sont arrangés dans différentes colonnes selon le nombre de COLUMNBREAKs dans une ligne. Le nombre de colonnes peut varier d'une ligne à l'autre.

La Figure 42 représente un exemple de disposition qui contient trois lignes avec un nombre de colonnes différent. La 1re et la 3e ligne ont 1 colonne, tandis que la 2e ligne a 2 colonnes de largeurs différentes. La Figure 41 représente le code source correspondant.

```
MENU menu_Layout_With_ImageColumn
{
       LABEL  "Layout with 'small Image' Column";
       STYLE DIALOG;
       ITEMS
       {
              "The quick brown fox jumps over the lazy dog. - "
              "The quick brown fox jumps over the lazy dog. - "
              "The quick brown fox jumps over the lazy dog.",
              ROWBREAK,
              var_DATE,
              var_ASCII,
              var_FLOAT,
              var_ENUM_COLOR,
              COLUMNBREAK,
              image_RED     (INLINE),
              image_YELLOW  (INLINE),
              image_GREEN   (INLINE),
              image_BLUE    (INLINE)
              ROWBREAK,
              method_ResetValues
       }
}
```

Figure 41 – Code source EDD pour une disposition avec petites images en ligne

L'intégralité de la largeur est déterminée par le contenu de la 2e ligne, qui compte deux colonnes. La largeur de la 1re colonne est déterminée en additionnant la largeur:

- du libellé le plus large, qui est "ENUMERATED";
- de l'élément de menu le plus large, qui est "var_ENUM_COLOR" et qui contient un élément d'énumération extra-large;
- de l'unité la plus large, qui est "mV";
- du statut de la variable.

La largeur de la 2e colonne est déterminée par les petites images en ligne.

Le contenu de la 1re ligne (chaîne constante) et de la 3e ligne (bouton) est aligné et étendu de manière à couvrir la même largeur que la 2e ligne. Ainsi, les champs d'entrée de la 1re et de la 3e ligne sont plus larges que les largeurs maximales respectives (voir Tableau 23).

En raison de la flexibilité de la structure de la disposition prévue par la disposition de largeur optimale des colonnes, il est possible de placer les petites images en ligne près de la 1re colonne, par exemple pour mettre en évidence une relation particulière entre ces colonnes.

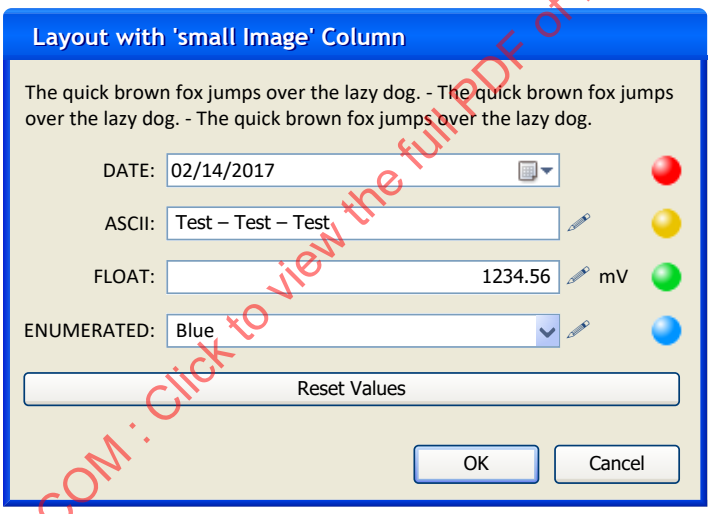


Figure 42 – Exemple de disposition avec petites images en ligne

La Figure 44 représente un exemple de disposition contenant 3 lignes avec un nombre de colonnes différent. La 1re ligne contient 3 colonnes, la 2e ligne contient un groupe imbriqué avec 2 colonnes, et la 3e ligne contient des variables couvrant un nombre de colonnes différent en définissant l'attribut WIDTH. La Figure 43 représente le code source correspondant.

```
MENU menu_Layout_MultiColumn_With_Group

        LABEL  "Multi-column Layout";
        STYLE DIALOG;
        ITEMS
        {
                var_INTEGER,
                COLUMNBREAK,
                var_DOUBLE,
                COLUMNBREAK,
                var_DATE,
                ROWBREAK,
                menu_Group_2C,
                ROWBREAK,
                var_ASCII,                  // Variable ASCII sans WIDTH → par défaut
                var_ASCII_Width_SMALL,             // Variable ASCII avec 'WIDTH SMALL'
                var_ASCII_Width_MEDIUM             // Variable ASCII avec 'WIDTH MEDIUM'
        }
}

MENU menu_Group_2C
{
        LABEL  "Group - 2C";               // GROUP avec 2 colonnes
        STYLE  GROUP;
        ITEMS
        {
                var_ASCII,
                var_ENUMERATED,
                COLUMNBREAK,
                var_FLOAT,
                var_INTEGER
        }
}
```

Figure 43 – Code source EDD pour une disposition pour plusieurs colonnes avec GROUP

La 1re ligne a 3 colonnes par COLUMNBREAKs, et les éléments de la 3e ligne couvrent jusqu'à 3 colonnes par attribut WIDTH. Par conséquent, les éléments de ces lignes sont alignés.

La largeur de la 1re colonne est déterminée en additionnant la largeur:

- du libellé le plus large, qui est "ASCII (MEDIUM = 3C)" dans la 3e ligne;
- du champ d'entrée (valeur) le plus large, qui est "var_ASCII" dans la 3e ligne;
- du statut de la variable.

La largeur de la 2e colonne est déterminée en additionnant la largeur:

- du libellé "DOUBLE" dans la 1re ligne;
- du champ d'entrée (valeur), qui est "var_DOUBLE" dans la 1re ligne;
- du statut de la variable;
- de l'unité "L/h".

La largeur de la 3e colonne est déterminée en additionnant la largeur:

- du libellé "DATE" dans la 1re ligne;
- du champ d'entrée (valeur), qui est "var_DATE" dans la 1re ligne.

La largeur de la disposition qui en résulte est déterminée en additionnant la largeur de la 1re, de la 2e et de la 3e colonne. La variable "var_ASCII_Width_SMALL" dans la 3e ligne est étendue afin de couvrir 2 colonnes; la variable "var_ASCII_Width_MEDIUM" est étendue afin de couvrir 3 colonnes et alignée. Les champs d'entrée respectifs sont alignés, ce qui est mis en évidence par les lignes auxiliaires de la Figure 44.

En raison des ROWBREAKs, le contenu du GROUP de la 2e est en premier lieu calculé indépendamment avec 2 colonnes. Les 2 colonnes du GROUP sont alors étendues pour couvrir la même largeur que les autres lignes.

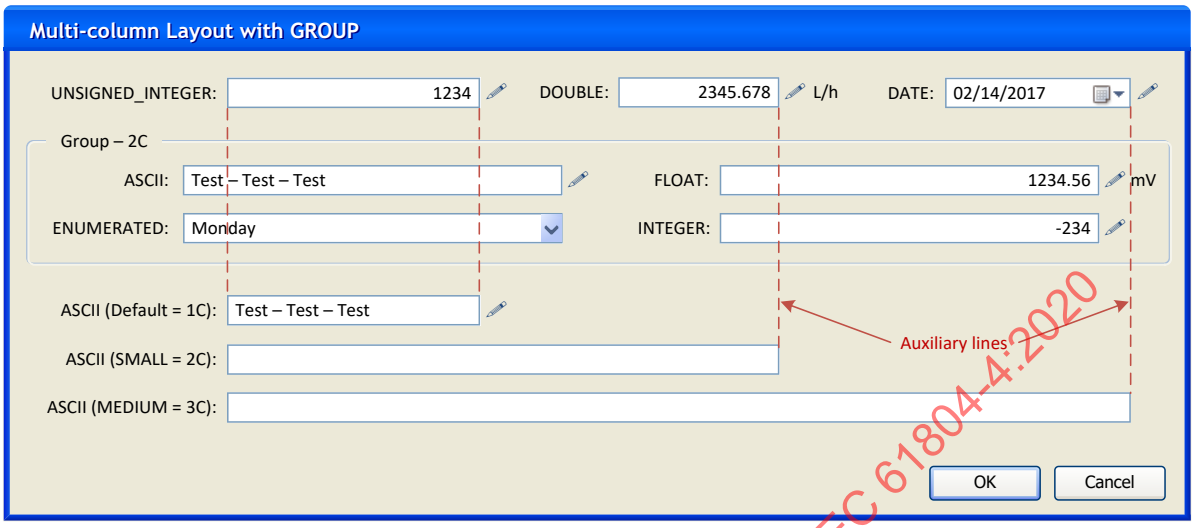


Figure 44 – Exemple de disposition pour plusieurs colonnes avec GROUP

4.7.5.6 Graphiques et diagrammes en ligne

Les graphiques et diagrammes peuvent apparaître dans les colonnes, tout comme les variables et les méthodes (voir Figure 45 et Figure 46).

```
MENU overview_page
{
    LABEL "Overview";
    STYLE PAGE;
    ITEMS
    {
        primary_variable,          /* variable */
        upper_range_value,         /* variable */
        lower_range_value,         /* variable */
        COLUMNBREAK,               /* saut de colonne */
        pv_graph                   /* graphique */
    }
}
```

Figure 45 – Exemple d'EDD pour les graphiques et diagrammes en ligne

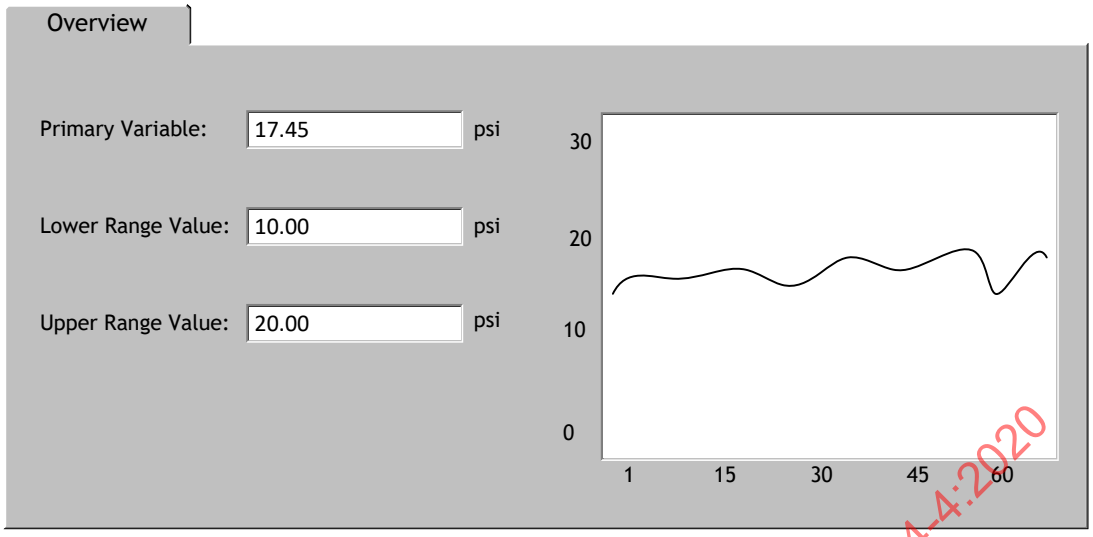


Figure 46 – Exemple d'application EDD pour un graphique en ligne

Si un graphique ou un diagramme possède un attribut WIDTH équivalent à XX_SMALL, X_SMALL, SMALL ou MEDIUM, il doit être affiché dans la colonne. Lorsque la valeur WIDTH équivaut à LARGE, X_LARGE ou XX_LARGE, la description indiquée en 4.7.5.7 doit s'appliquer.

4.7.5.7 Graphiques et diagrammes pleine largeur

L'exemple suivant représente la disposition de graphiques et de diagrammes couvrant plusieurs colonnes pour chaque type de disposition. Le code source EDD correspondant est représenté à la Figure 47.

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

MENU overview_page
{
    LABEL "Overview";
    STYLE PAGE;
    ITEMS
    {
        primary_variable,           /* variable */
        COLUMNBREAK,                /* saut de colonne */
        lower_range_value,          /* variable */
        COLUMNBREAK,                /* saut de colonne */
        upper_range_value,          /* variable */
        pv_graph,                   /* graphique avec WIDTH X_LARGE, occupe 5 colonnes */
        ROWBREAK,                   /* saut de ligne */
        reload_pv_graph,            /* méthode */
        COLUMNBREAK,                /* saut de colonne */
        save_pv_graph               /* méthode */
    }
}
```

Figure 47 – Exemple d'EDD pour les graphiques et diagrammes pleine largeur

Disposition de largeur égale des colonnes:

Si un graphique ou un diagramme possède un attribut WIDTH équivalent à X_LARGE ou XX_LARGE, il couvre la largeur de la fenêtre, de la boîte de dialogue, de la page ou du groupe. Il peut exister des éléments supplémentaires avant et après le graphique. Dans ce cas, les éléments précédant le graphique ou diagramme sont divisés en un groupe de colonnes dans une ligne séparée et les éléments suivant le graphique ou diagramme sont divisés en un autre groupe de colonnes dans une ligne séparée.

La Figure 48 représente la disposition de largeur égale des colonnes qui en résulte.

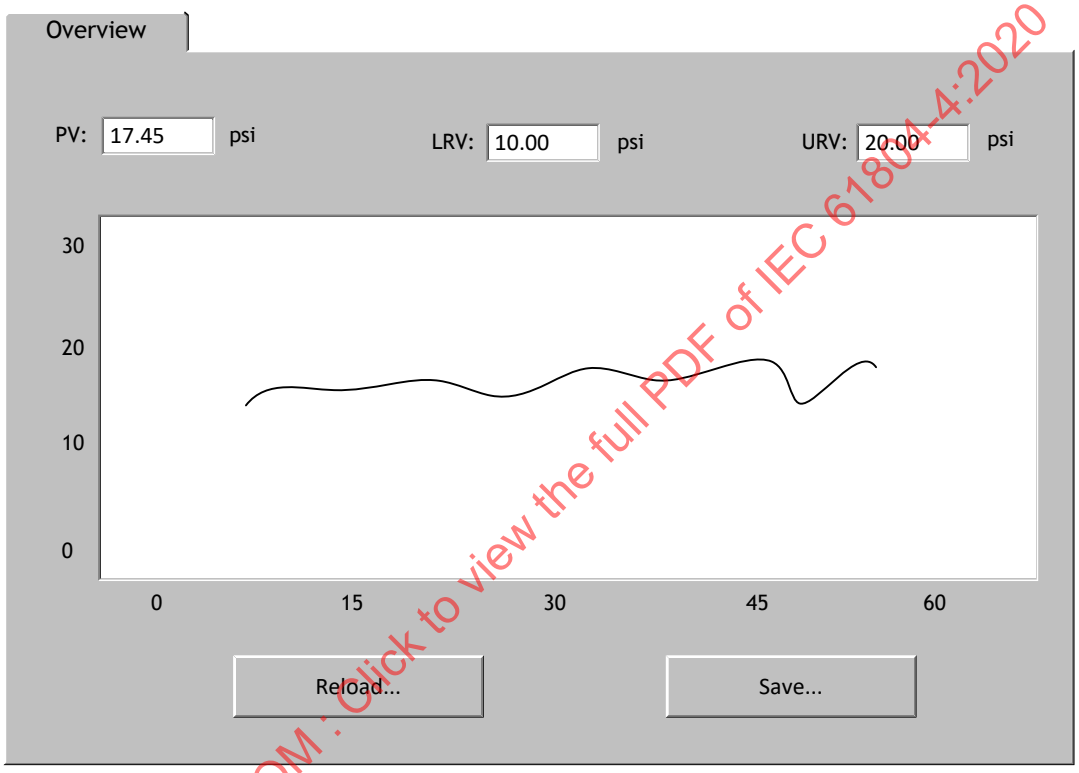


Figure 48 – Exemple d'application EDD pour un graphique pleine largeur dans une disposition avec des colonnes de largeur égale

NOTE Il y a trois colonnes avant le graphique et deux colonnes après celui-ci.

Disposition de largeur optimale des colonnes:

Conformément au code source EDD représenté à la Figure 47, la 1re ligne contient 3 colonnes, la 1re et la 2e colonne avec un élément de menu chacune et la 3e colonne avec 2 éléments de menu. Etant donné que des colonnes peuvent atteindre une colonne supérieure à la 5e, le graphique est restitué en ligne à partir de la 3e colonne sans être déplacé vers la ligne suivante par un ROWBREAK implicite. La Figure 49 représente la disposition de largeur optimale des colonnes qui en résulte.

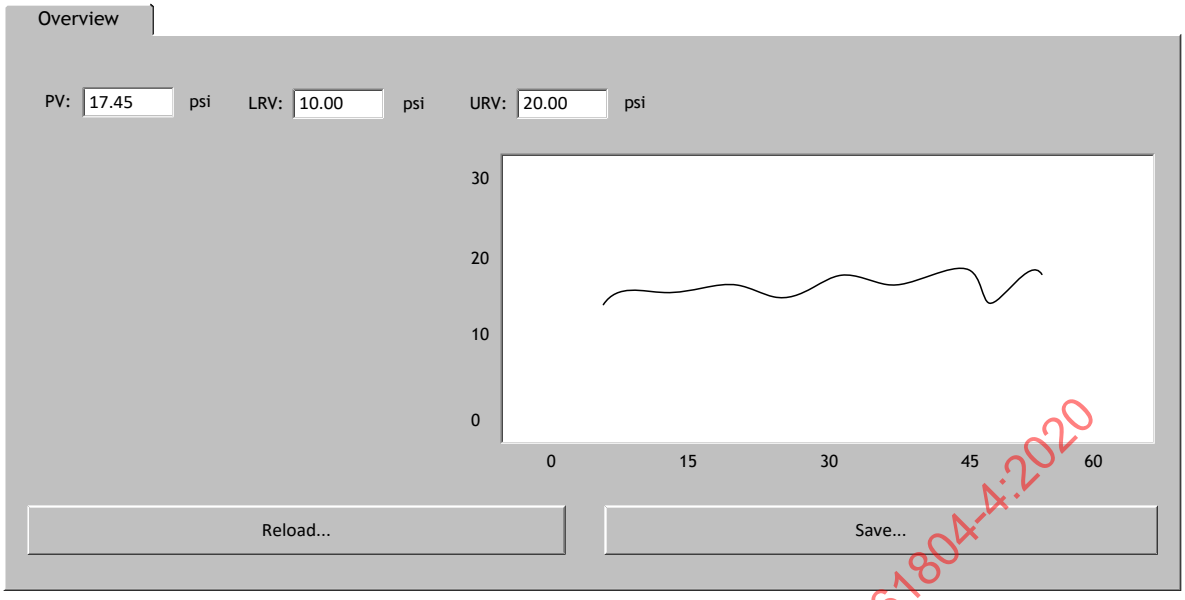


Figure 49 – Exemple d'application EDD pour un graphique pleine largeur dans une disposition avec des colonnes de largeur optimale

S'il convient de restituer le graphique à partir de la 1re colonne, le développeur EDD doit insérer un ROWBREAK avant le graphique (pv_graph).

4.7.5.8 Conteneurs imbriqués

Un conteneur peut être imbriqué dans d'autres conteneurs. La Figure 50 et la Figure 51 représentent une page imbriquée dans une fenêtre et deux groupes imbriqués dans la page.

```
MENU overview_window
{
    LABEL "Device Overview";
    STYLE WINDOW;
    ITEMS
    {
        overview_page                          /* page */
    }
}
MENU overview_page
{
    LABEL "Overview";
    STYLE PAGE;
    ITEMS
    {
        range_values,                      /* groupe */
        sensor_limits,                     /* groupe */
        COLUMNBREAK,                       /* saut de colonne */
        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 50 – Exemple d'EDD pour les conteneurs imbriqués

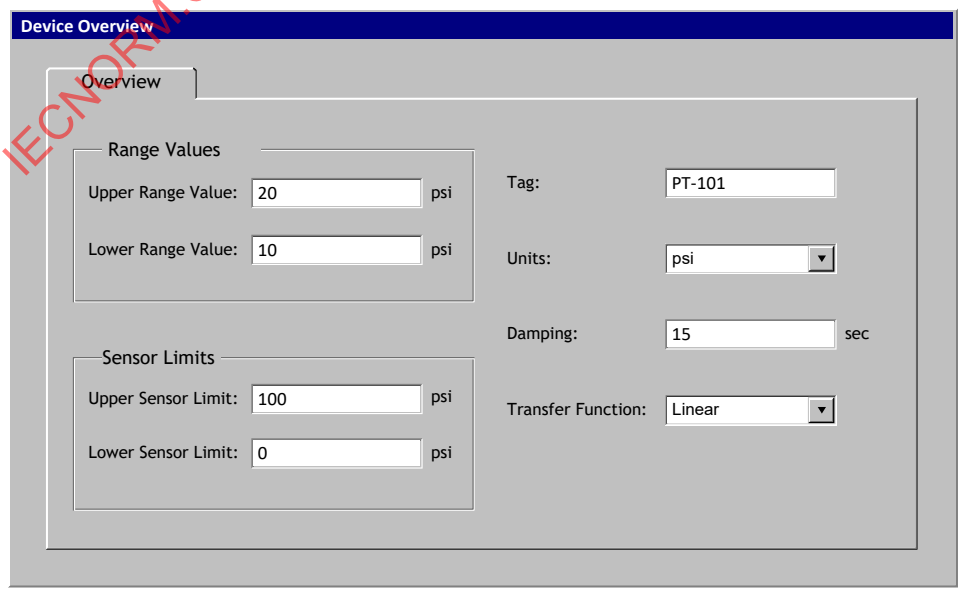


Figure 51 – Exemple d'application EDD pour les conteneurs imbriqués

4.7.5.9 Affichages d'édition

Les EDIT_DISPLAYS peuvent être référencées dans des conteneurs. Lorsqu'un affichage d'édition apparaît dans un conteneur, il doit être représenté sous forme de bouton ou de lien hypertexte. Lorsque le bouton est activé, une boîte de dialogue qui affiche le contenu de l'affichage d'édition doit apparaître (voir Figure 52 et Figure 53). Les boutons OK et Cancel ne font pas partie du domaine d'application de la présente spécification.

```
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                        /* édition d'affichage */
    }
}
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 52 – Exemple d'EDD pour les éléments EDIT_DISPLAY

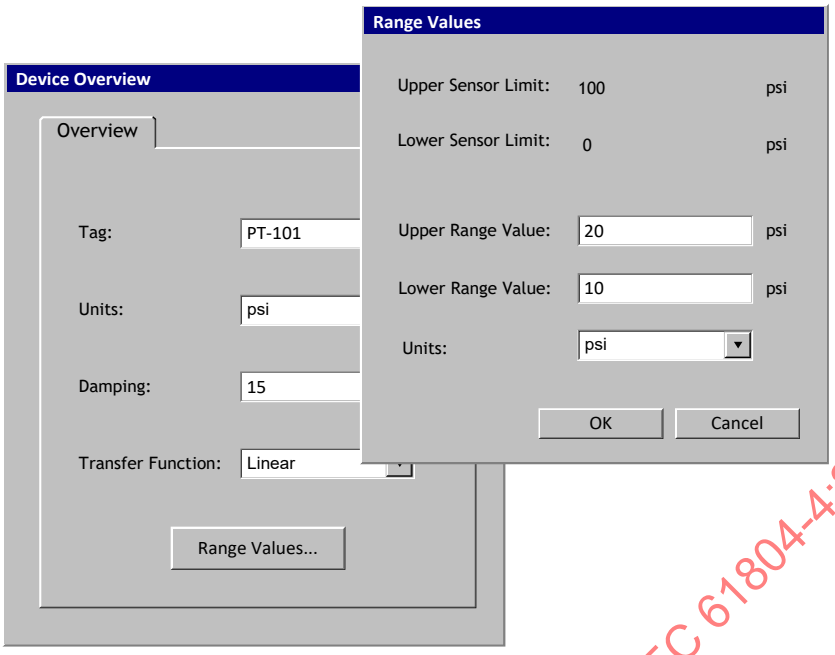


Figure 53 – Exemple d'application EDD pour les éléments EDIT_DISPLAY

4.7.5.10 Images

Les images peuvent également être placées dans des conteneurs. Si une image est référencée par un qualificatif d'élément de menu INLINE, l'image s'affiche dans la colonne actuelle du conteneur (voir Figure 54 et Figure 55). Sinon, l'image s'affiche sur la largeur du conteneur.

```
MENU overview_page
{
    LABEL "Overview";
    STYLE PAGE;
    ITEMS
    {
        voltage (ALIGN_LEFT), /* Référence à une image */
        ramp_start,           /* variable */
        ramp_end,             /* variable */
        COLUMNBREAK,
        logo(INLINE)          /* Référence à une image affichée en ligne */
    }
}
```

Figure 54 – Exemple d'EDD pour les images

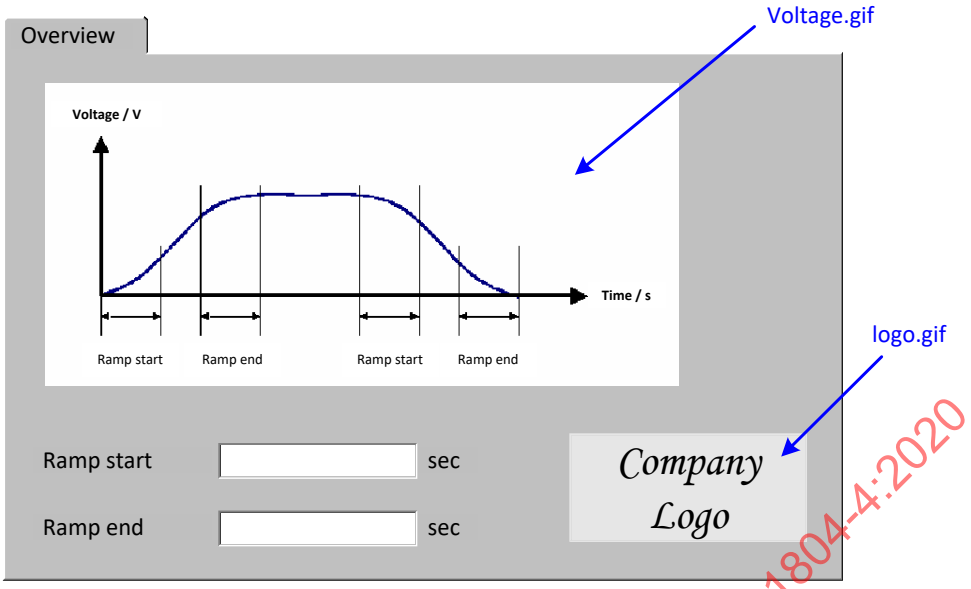


Figure 55 – Exemple d'application EDD pour les images

Disposition de largeur optimale des colonnes:

Dans une disposition avec largeur de colonne optimale, l'image ne doit pas être réduite, même si INLINE est spécifié. La Figure 57 représente un exemple de disposition avec une image large restituée en ligne. La Figure 56 représente le code source EDD correspondant.

```
MENU overview_page
{
    LABEL "Overview";
    STYLE PAGE;
    ITEMS
    {
        ramp_start,          /* variable */
        ramp_end,            /* variable */
        COLUMNBREAK,
        voltage (INLINE)     /* Référence à une image affichée en ligne */
    }
}
```

Figure 56 – Exemple d'EDD pour de larges images en ligne

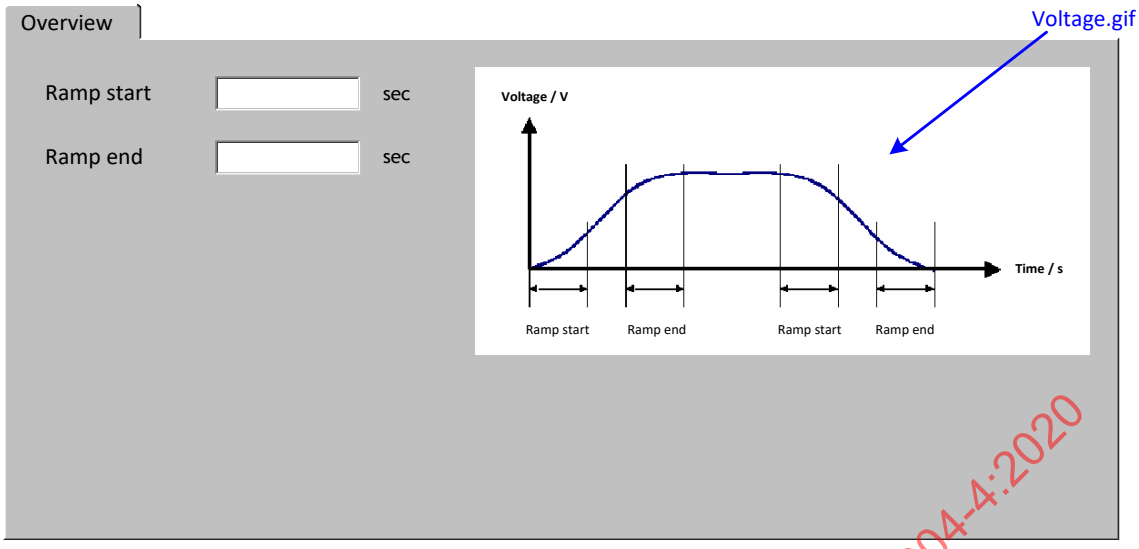


Figure 57 – Exemple de disposition avec une image large en ligne

4.7.6 Interface utilisateur conditionnelle

4.7.6.1 Vue d'ensemble

Pour contrôler l'apparence des éléments d'interface utilisateur, l'EDDL dispose de plusieurs possibilités:

- l'attribut VISIBILITY;
- les expressions conditionnelles du MENU ITEM;
- l'attribut VALIDITY.

4.7.6.2 VISIBILITY

L'attribut VISIBILITY des constructions de base (par exemple, MENU, VARIABLE, IMAGE, GRAPH, CHART, GRID) permet de contrôler l'apparence des éléments d'interface utilisateur. Les valeurs TRUE ou FALSE de l'attribut VISIBILITY sont destinées à être évaluées de manière dynamique en fonction des valeurs de VARIABLE.

Dans le cas d'un MENU, l'attribut VISIBILITY du MENU enfant n'affecte pas la disposition d'écran du MENU. Dans la disposition d'écran du MENU, le même espace que si le MENU enfant était affiché est réservé.

4.7.6.3 Expressions conditionnelles dans MENU ITEM

Les expressions conditionnelles de la liste MENU ITEM permettent de contrôler la disposition d'écran. Dans la disposition d'écran, aucun espace n'est réservé si un élément n'est pas affiché.

4.7.6.4 VALIDITY

L'attribut VALIDITY des constructions de base (par exemple, MENU, VARIABLE, IMAGE, GRAPH, CHART, GRID) permet de contrôler l'apparence des éléments d'interface utilisateur. Les valeurs TRUE ou FALSE de l'attribut VALIDITY sont destinées à être évaluées de manière dynamique en fonction des valeurs de VARIABLE.

Pour les sessions en ligne et hors ligne, si une construction VALIDITY est évaluée de TRUE à FALSE, l'application EDD doit empêcher la construction de s'afficher.

Pour les sessions en ligne et hors ligne, si une construction VALIDITY est évaluée de FALSE à TRUE, l'application EDD doit afficher la construction.

Pour une session en ligne, lorsque l'attribut VALIDITY d'une VARIABLE est évalué de FALSE à TRUE, l'application EDD doit afficher la valeur de la VARIABLE comme étant non initialisée (voir Tableau 20), jusqu'à ce que la VARIABLE soit lue à partir de l'appareil ou modifiée pendant la même session d'édition.

Pour une session hors ligne, lorsque l'attribut VALIDITY d'une VARIABLE est évalué de FALSE à TRUE, l'application EDD doit afficher la valeur de la VARIABLE à partir du cache actuel.

Il convient que les VARIABLES avec un attribut VALIDITY FALSE référencé depuis un MENU ne soient pas lues à partir de l'appareil.

L'attribut VALIDITY affecte la disposition d'écran. Dans la disposition d'écran, aucun espace n'est réservé si un élément n'est pas valide. Les éléments de données d'une GRID pour lesquels VALIDITY = FALSE doivent se comporter de la même façon qu'avec VISIBILITY = FALSE (c'est-à-dire que l'espace est réservé, mais que la cellule est vide).

La Figure 58 donne un exemple d'EDD pour VALIDITY dans une session en ligne. Le Tableau 25, le Tableau 26, le Tableau 27 et le Tableau 28 montrent le fonctionnement d'une application EDD avec VALIDITY dans une session en ligne.

```
VARIABLE option
{
        LABEL "Option";
        HANDLING READ & WRITE;
        TYPE ENUMERATED
        {
                {0, "Disable"},
                {1, "Internal Option"},
                {2, "External Option"}
        }
}

VARIABLE internal
{
        LABEL "Internal";
        VALIDITY IF (option == 1) {TRUE;} ELSE {FALSE;}
        HANDLING READ & WRITE;
        TYPE FLOAT
        {
                DISPLAY_FORMAT ".1f";
                EDIT_FORMAT "8.1f";
        }
}

VARIABLE external
{
        LABEL "External";
        VALIDITY IF (option == 2) {TRUE;} ELSE {FALSE;}
        HANDLING READ & WRITE;
        TYPE FLOAT
        {
                DISPLAY_FORMAT ".1f";
                EDIT_FORMAT "8.1f";
        }
}

MENU selection_option
{
        LABEL "Selection";
        STYLE DIALOG;
        ITEMS
        {
                option (DISPLAY_VALUE),
                internal (DISPLAY_VALUE),
                external (DISPLAY_VALUE)
        }
}
```

Figure 58 – Exemple d'EDD pour VALIDITY dans une session en ligne

Tableau 25 – Exemple 1 pour VALIDITY dans une session en ligne

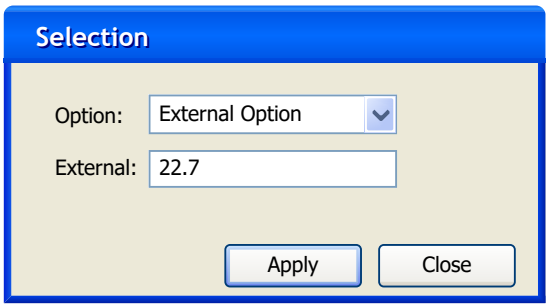
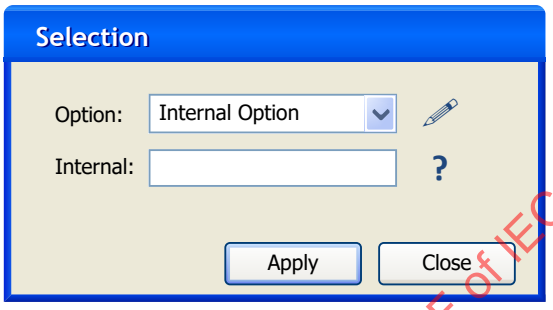
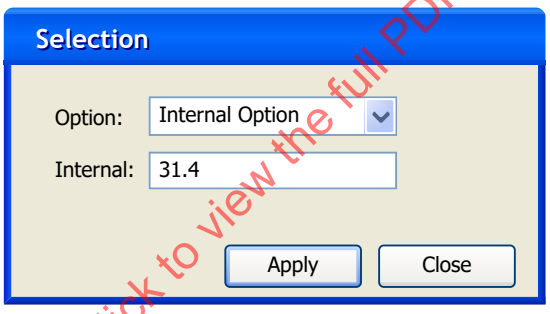
Etape	Interface utilisateur	Description
Etape 1: "Option" d'état initial = "External Option"		"Internal" est VALIDITY FALSE et n'est pas affiché. "External" est VALIDITY TRUE et est affiché.
Etape 2: Edition "Option" = "Internal Option"		"External" devient VALIDITY FALSE. "External" est retiré de l'affichage. "Internal" devient VALIDITY TRUE. "Internal" est affiché comme étant éditable et vide avec indication de l'état "uncertain" proche du champ de valeur.
Etape 3: Appuyer sur "Apply" pour envoyer la valeur modifiée		"Option" est envoyé "Internal" n'est pas envoyé. "Internal" est lu et affiché.

Tableau 26 – Exemple 2 pour VALIDITY dans une session en ligne

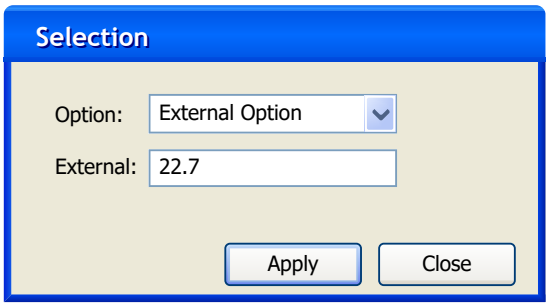
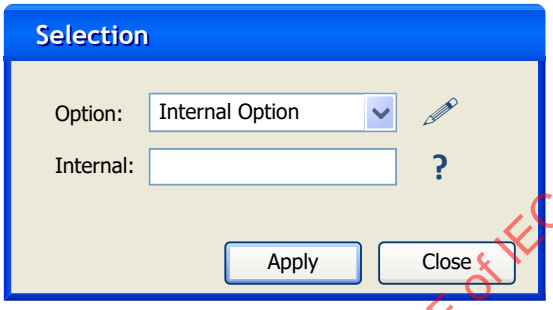
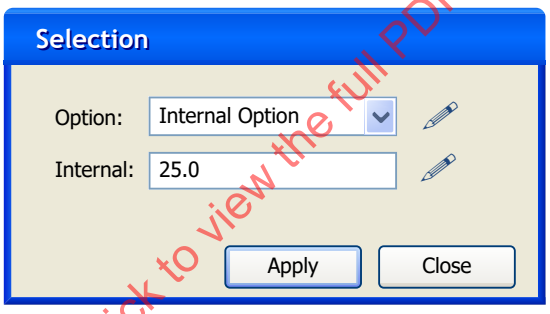
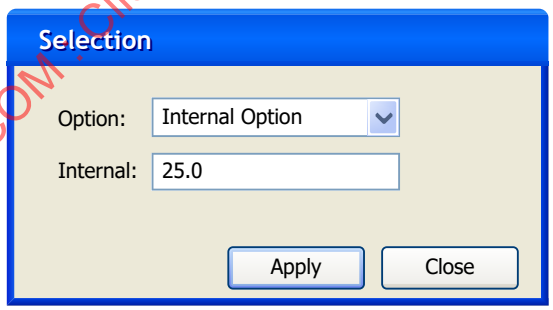
Etape	Interface utilisateur	Description
Etape 1: "Option" d'état initial = "External Option"		"Internal" est VALIDITY FALSE et n'est pas affiché. "External" est VALIDITY TRUE et est affiché.
Etape 2: Edition 2 "Option" = "Internal Option"		"External" devient VALIDITY FALSE. "External" est retiré de l'affichage. "Internal" devient VALIDITY TRUE. "Internal" est affiché comme étant éditable et vide avec indication de l'état "uncertain" proche du champ de valeur.
Etape 3: Edition de "Internal" à 25.0		La valeur éditée est affichée avec indication de l'état "changed" proche du champ de valeur.
Etape 4: Appuyer sur "Apply" pour envoyer la valeur modifiée		"Option" est envoyé. "Internal" est envoyé.

Tableau 27 – Exemple 3 pour VALIDITY dans une session en ligne

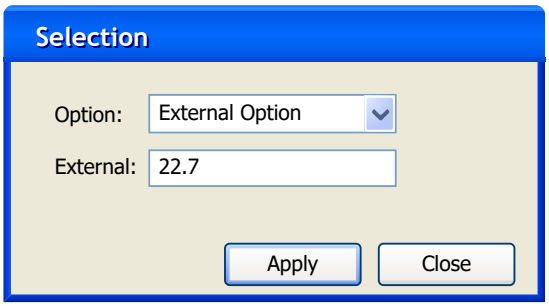
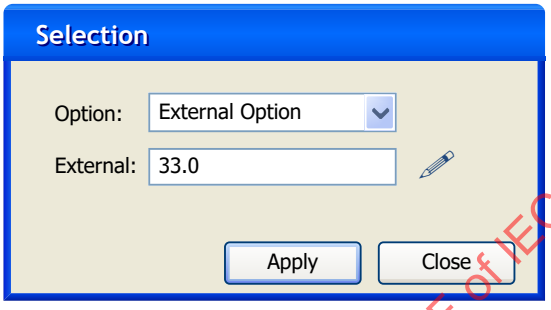
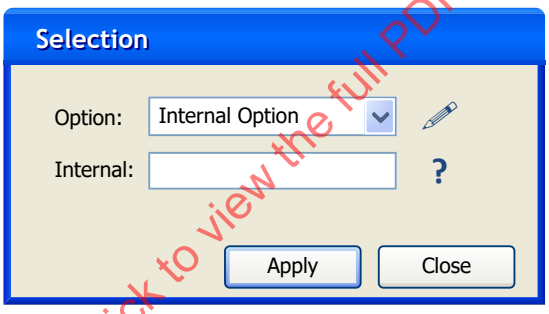
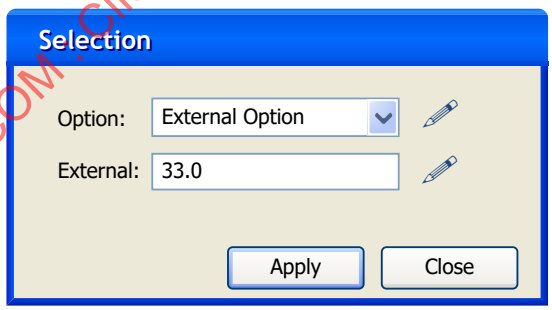
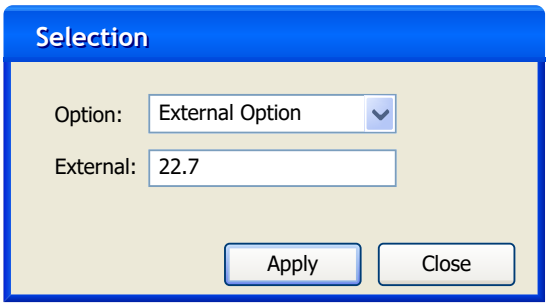
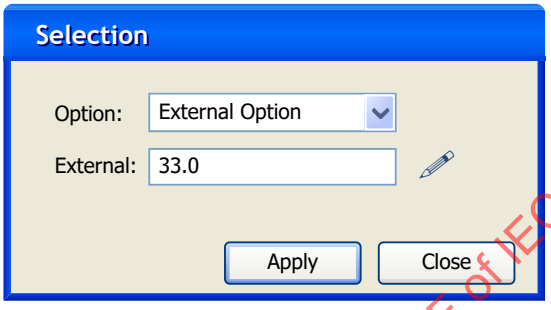
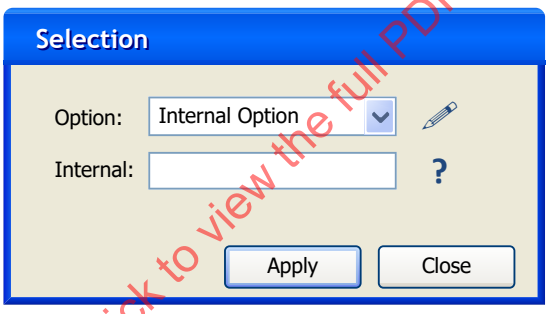
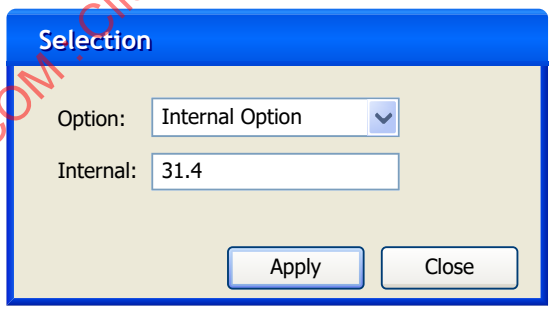
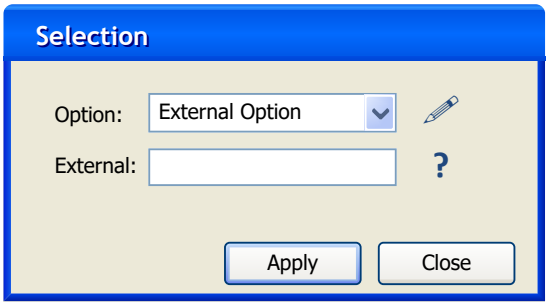
Etape	Interface utilisateur	Description
Etape 1: "Option" d'état initial = "External Option"		"Internal" est VALIDITY FALSE et n'est pas affiché. "External" est VALIDITY TRUE et est affiché.
Etape 2: Edition de "External" à 33.0		La valeur éditée est affichée avec indication de l'état "changed" proche du champ de valeur.
Etape 3: Rendre "External" VALIDITY FALSE en éditant "Option" qui devient "Internal Option"		"External" devient VALIDITY FALSE. "External" est retiré de l'affichage.
Etape 4: Rendre "External" VALIDITY TRUE à nouveau en éditant "Option" qui devient "External Option"		"External" devient VALIDITY TRUE. "External" s'affiche avec la valeur éditée précédente avec indication de l'état "changed" proche du champ de valeur.

Tableau 28 – Exemple 4 pour VALIDITY dans une session en ligne

Etape	Interface utilisateur	Description
Etape 1: "Option" d'état initial = "External Option"		"Internal" est VALIDITY FALSE et n'est pas affiché. "External" est VALIDITY TRUE et est affiché.
Etape 2: Edition de "External" à 33.0		La valeur éditée est affichée avec indication de l'état "changed" proche du champ de valeur.
Etape 3: Rendre "External" VALIDITY FALSE en éditant "Option" qui devient "Internal Option"		"External" devient VALIDITY FALSE. "External" est retiré de l'affichage.
Etape 4: Appuyer sur "Apply" pour envoyer la valeur modifiée		"Option" est envoyé. "Internal" n'est pas envoyé. "Internal" est lu et affiché. "External" précédemment édité, mais VALIDITY FALSE n'est pas envoyé.
Etape 5: Rendre "External" VALIDITY TRUE à nouveau en éditant "Option" qui devient "External Option"		"External" devient VALIDITY TRUE. "External" est affiché comme étant éditable et vide avec indication de l'état "uncertain" proche du champ de valeur.

4.8 Eléments graphiques

Les appareils de terrain équipés d'un microprocesseur intelligent deviennent de plus en plus sophistiqués et complexes. Dans certains cas, les mesures ou contrôles qui étaient auparavant irréalisables ou qui exigeaient de nombreux équipements ont été intégrés en un seul appareil. L'EDDL prend en charge ces appareils très sophistiqués.

Ces constructions peuvent évidemment être utilisées dans de nombreux autres types d'appareils. Le développeur EDD dispose d'un contrôle total sur le contenu de l'EDD. Il lui incombe de décider quelles constructions EDDL doivent être utilisées.

Afin de répondre aux besoins de ces appareils complexes, la prise en charge des capacités suivantes a été ajoutée:

- élaboration de graphiques;
- élaboration de diagrammes;
- gestion du contenu des images;
- gestion des données tabulaires.

L'EDDL prend en charge deux types de représentations des données graphiques. Il s'agit des représentations GRAPH et CHART. Un CHART est utilisé pour afficher les valeurs réelles et un GRAPH sert à afficher les données conservées qui peuvent être lues à partir de l'appareil ou d'une mémoire permanente. Il est fréquent de comparer une forme d'onde de l'appareil à une forme d'onde de référence ou à la forme d'onde d'une date/opération spécifique conservée par l'application EDD. La principale différence entre ces deux constructions est la suivante: l'application EDD gère les données d'un CHART, et pour un GRAPH, l'EDD définit, lit et met à jour elle-même les données. Les méthodes de gestion des données sont facultatives pour les CHARTs, mais généralement nécessaires pour les GRAPHs.

Un GRAPH contient une liste de WAVEFORMs tracées conjointement. Le GRAPH peut être référencé depuis un MENU ou dans le cadre d'une METHOD. Les WAVEFORMs disposent d'un nombre minimal d'attributs obligatoires pour permettre au développeur EDD de produire facilement un graphique. Pour les développeurs souhaitant davantage de contrôle, une large gamme d'attributs facultatifs est disponible (taille d'affichage, axe, points clés, etc.).

Ces constructions sont particulièrement utiles pour le tracé des signatures de vannes et des signaux radar. Les données de l'appareil de terrain et celles conservées par l'application EDDL peuvent être tracées sur le même GRAPH. Par exemple, les données de l'appareil de terrain peuvent être spécifiées dans une WAVEFORM et les données permanentes d'un FILE dans une autre WAVEFORM. Les deux WAVEFORMs peuvent être tracées sur le même GRAPH.

Un GRAPH est un instantané de l'appareil ou des propriétés du processus, alors qu'un CHART apporte une vue d'une tendance évoluant avec le temps. Un CHART possède des SOURCEs qui sont échantillonnées régulièrement. Les tracés de style bandes, balayage et domaine d'application sont pris en charge afin de représenter les tendances au fil du temps. Les types horizontal, vertical et mesureur permettent une représentation graphique d'une valeur instantanée unique. Des régions (bandes) colorées peuvent être créées en utilisant les lignes de limites (c'est-à-dire, les types de lignes LOW_LOW_LIMIT, LOW_LIMIT, HIGH_LIMIT et HIGH_HIGH_LIMIT).

Pour les lignes de limites du même type de ligne partageant la même ordonnée, les lignes de limites doivent être triées par valeur. La liste de lignes de limites ordonnée qui en résulte doit créer des régions (bandes) successives sur des CHARTs de type instrument de mesure. Pour les types de limites basses, la limite supérieure d'une région limite doit être fixée par la valeur de la ligne de limite correspondante. Pour les types de limites hautes, la limite inférieure d'une région limite doit être fixée par la valeur de la ligne de limite correspondante. La couleur d'une région limite doit être la LINE_COLOR de la ligne de limite correspondante. La limite inférieure de la première région limite doit commencer à la valeur minimale de l'axe. La limite supérieure de la première région limite doit atteindre la valeur maximale de l'axe.

La Figure 59 représente un exemple de régions limites restituées par une application selon la définition EDD représentée à la Figure 60.

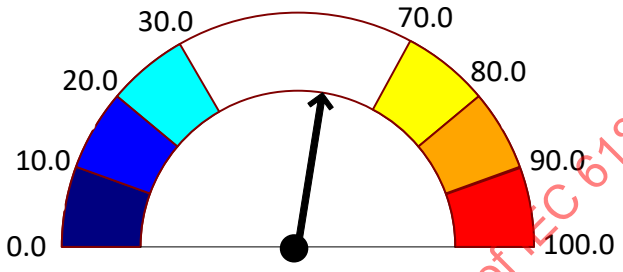


Figure 59 – Exemple d'application EDD pour un mesureur avec régions limites

```
VARIABLE local_minlow_limit
{
   CLASS LOCAL;
   TYPE FLOAT;
   DEFAULT_VALUE 10.0;
}

VARIABLE local_midlow_limit
{
   CLASS LOCAL;
   TYPE FLOAT;
   DEFAULT_VALUE 20.0;
}

VARIABLE local_maxlow_limit
{
   CLASS LOCAL;
   TYPE FLOAT;
   DEFAULT_VALUE 30.0;
}

VARIABLE local_minhi_limit
{
   CLASS LOCAL;
   TYPE FLOAT;
   DEFAULT_VALUE 70.0;
}

VARIABLE local_midhi_limit
{
   CLASS LOCAL;
   TYPE FLOAT;
   DEFAULT_VALUE 80.0;
}

VARIABLE local_maxhi_limit
{
   CLASS LOCAL;
   TYPE FLOAT;
   DEFAULT_VALUE 90.0;
}

AXIS gaugeAxis
{
   MIN_VALUE deviceVariables[0].LOWER_SENSOR_LIMIT;  //  0.0 Pour cet exemple
   MAX_VALUE deviceVariables[0].UPPER_SENSOR_LIMIT;  //100.0 Pour cet exemple
}

SOURCE minlowSource
{
   LINE_TYPE LOW_LIMIT;
   LINE_COLOR NAVY;
   Y_AXIS gaugeAxis;
   MEMBERS
   {
      LIMIT, local_minlow_limit;
   }
}

SOURCE midlowSource
{
   LINE_TYPE LOW_LIMIT;
   LINE_COLOR BLUE;
   Y_AXIS gaugeAxis;
   MEMBERS
   {
      LIMIT, local_midlow_limit;
   }
}

SOURCE maxlowSource
{
   LINE_TYPE LOW_LIMIT;
   LINE_COLOR AQUA;
   Y_AXIS gaugeAxis;
   MEMBERS
   {
      LIMIT, local_maxlow_limit;
```

```
    }
}

SOURCE minhiSource
{
   LINE_TYPE HIGH_LIMIT;
   LINE_COLOR YELLOW;
   Y_AXIS gaugeAxis;
   MEMBERS
   {
      LIMIT, local_minhi_limit;
   }
}

SOURCE midhiSource
{
   LINE_TYPE HIGH_LIMIT;
   LINE_COLOR ORANGE;
   Y_AXIS gaugeAxis;
   MEMBERS
   {
      LIMIT, local_midhi_limit;
   }
}

SOURCE maxhiSource
{
   LINE_TYPE HIGH_LIMIT;
   LINE_COLOR RED;
   Y_AXIS gaugeAxis;
   MEMBERS
   {
      LIMIT, local_maxhi_limit;
   }
}

CHART exampleGauge
{
   LABEL "Example Gauge";
   TYPE GAUGE;
   MEMBERS
   {
      GAUGE_VALUE,  valueSource;
      MINLOW_LIMIT, minlowSource;
      MIDLOW_LIMIT, midlowSource;
      MAXLOW_LIMIT, maxlowSource;
      MINHI_LIMIT,  minhiSource;
      MIDHI_LIMIT,  midhiSource;
      MAXHI_LIMIT,  maxhiSource;
   }
}
```

Figure 60 – Exemple d'EDD pour un mesureur avec régions limites

L'état et la performance de certains appareils ne peuvent pas être déterminés empiriquement. Pour ces appareils, la performance relative est l'indicateur de l'état de l'appareil. La construction FILE permet d'accéder aux données conservées dans l'appareil par l'application EDDL. L'EDD spécifie les données, qui doivent être stockées avec une syntaxe semblable à celle de la COLLECTION. Le développeur EDD définit les données à stocker dans les combinaisons de VARIABLEs, d'ARRAYs, de COLLECTIONs ou de LISTs.

La construction LIST a été ajoutée afin d'améliorer la flexibilité du langage lors de la spécification du stockage des données permanentes dans un FILE. La construction LIST est une matrice de longueur variable. Des données peuvent être ajoutées à la liste ou supprimées de celle-ci au fil du temps. Chaque entrée de la LIST est du même type que celui spécifié par le développeur EDD.

La construction COLLECTION prend en charge toutes les combinaisons des types de membres. Auparavant, les collections étaient seulement autorisées à contenir des références d'un seul type (VARIABLE, ARRAY, COLLECTION, MENU).

5 Description de données EDDL

5.1 Données d'appareil stockées dans l'application EDDL

5.1.1 Vue d'ensemble

Plusieurs types d'appareils complexes ont besoin d'accéder aux données de performance historiques afin d'évaluer l'état de fonctionnement actuel. Deux exemples représentent ces types d'appareils: les positionneurs de vanne et les mesureurs de niveau radar.

Dans le cas d'un positionneur de vanne, une signature de référence est utilisée lorsque l'installation est achevée. Cette signature est comparée à la signature réelle à une date ultérieure. Des différences significatives entre les signatures peuvent indiquer qu'il est exigé de procéder à la maintenance.

Dans le cas d'un mesureur radar, plusieurs signaux retour peuvent être enregistrés sous différentes conditions d'exploitation (par exemple, différents niveaux de réservoir ou différents équipements d'exploitation). L'examen et/ou la comparaison de ces signaux peuvent permettre l'optimisation du fonctionnement du mesureur ou la détection d'un changement imprévu dans les conditions d'exploitation.

Le développeur EDD peut spécifier les données devant être stockées par l'application EDDL. Ces données peuvent être rappelées à une date ultérieure aux fins d'évaluation de la performance et/ou de l'état de l'appareil.

5.1.2 FILE

La construction FILE permet à un développeur EDD de spécifier les données devant être stockées pour une utilisation ultérieure. A l'instar d'une COLLECTION, elle ne fait pas de différence entre le référencement de données direct ou l'utilisation de la référence de fichier. Les deux constructions assurent l'accès aux mêmes données. La différence entre les constructions FILE et COLLECTION est que les données référencées par des FILEs sont stockées.

Des données permanentes sont associées à une seule et même instance d'appareil. Envisager un système comportant quatre positionneurs de vanne identiques. La même EDD est utilisée pour les quatre appareils. Néanmoins, il existe quatre ensembles de données d'instance différents, un pour chaque positionneur de vanne. Lorsque la construction FILE est utilisée, l'application EDD conserve des données permanentes qui sont associées à chaque appareil. Si la construction FILE est utilisée pour conserver la signature de vanne de l'appareil, cette signature est alors disponible à une date ultérieure. La signature du positionneur de vanne n°2 n'est pas disponible lorsque l'EDD est utilisée par le positionneur de vanne n°1.

La construction FILE spécifie uniquement la structure des données à stocker. Elle ne spécifie pas la manière dont l'application EDD stocke les données, ni le moment où les données sont chargées dans la mémoire locale. L'application EDD peut charger les données du FILE au moment de l'instanciation de l'appareil ou peut fournir les données lorsque l'EDD en fait la demande. Les données permanentes peuvent être conservées dans une mémoire flash ou sur un disque dur, sur l'ordinateur exécutant l'application EDD ou sur un serveur de fichiers.

NOTE Les membres non valides d'un FILE n'invalident pas le FILE tout entier. Toutefois, les applications EDD reconnaissent lorsqu'un membre ou une partie d'un membre est non valide. Par exemple: Une application EDD peut simplement prendre un instantané des MEMBERS et identifier le membre non valide. Ensuite, cet élément EDD peut être initialisé afin d'invalider le FILE de temps suivant pour que cette instance d'appareil soit chargée par l'application EDD.

Le FILE peut être stocké sous la forme d'un fichier plat dans la structure de fichiers des systèmes d'exploitation ou peut être intégré à la base de données de l'application EDD. Dans tous les cas, l'accès à l'ensemble des données du FILE est disponible lorsque l'EDD est instanciée. L'EDD n'a pas besoin d'appeler des commandes d'ouverture, de fermeture, de lecture ou d'écriture des fichiers de niveau inférieur.

Le développeur EDD définit le schéma ou la structure de données du FILE. Le FILE comporte une liste de membres pouvant être des combinaisons d'éléments de données, par exemple VARIABLEs, ARRAYs, RECORDs, COLLECTIONs ou LISTs. Cette flexibilité permet au développeur EDD de stocker un ensemble fixe de données uniquement avec des VARIABLEs, des ARRAYs, des RECORDs et des COLLECTIONs. Le développeur EDD peut également stocker une quantité variable de données à l'aide de la construction LIST (conjointement aux constructions VARIABLE, ARRAY, RECORD et COLLECTION).

La Figure 61 est un exemple simple de déclaration FILE. Dans ce cas, une demande de prise en charge d'un fichier appelé "reference_valve_signature" est soumise à l'application EDDL.

```
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 61 – Exemple de déclaration de fichier

Ce FILE comporte une brève note ("valve_signature_comment") concernant la signature, et la signature proprement dite ("valve_stroke"). La signature est une matrice de 1 000 échantillons de position de vanne également espacés.

Il s'agit d'un exemple simple. Dans une application réelle, les échantillons de position peuvent ne pas être également espacés, ce qui nécessitera un temps d'échantillonnage. Si tel est le cas, une autre matrice de temps d'échantillonnage sera nécessaire. D'autres informations complémentaires (date, heure, opérateur, etc.) peuvent être ajoutées.

L'accès aux membres d'un FILE se fait à l'aide de la notation à point comme pour le référencement d'un membre de COLLECTION. Ainsi

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

accéderait à la 10e valeur dans la matrice de signature, ou valve_stroke[10]. Il est également simple de tracer cette signature et de la comparer à la signature réelle. Bien que cela soit normalement réalisé par une méthode, il est également possible de le faire par le biais d'un MENU.

La Figure 62 représente un GRAPH dans un 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 62 – Exemple de comparaison des signatures d'une vanne

Dans l'exemple de la Figure 62, les données échantillonnées lors d'une course récente de la vanne sont stockées dans l'attribut "current_stroke" et l'intervalle d'échantillonnage (l'inverse du taux d'échantillonnage) indique l'espacement temporel entre les points d'échantillonnage. Deux WAVEFORMs et un GRAPH sont déclarés, ainsi qu'un MENU qui contient le GRAPH. Le GRAPH est affiché par l'application EDDL lorsque le MENU est accédé.

Lorsque le graphique est affiché, l'application EDD reconnaît que l'attribut "valve_stroke" est un membre d'un FILE. Par conséquent, lorsque le GRAPH est affiché, il convient que les données soient automatiquement fournies à partir du FILE "reference_valve_signature".

5.1.3 LIST

La construction LIST est une matrice insérable de longueur variable. Chaque élément stocké dans la liste a exactement la même structure. Cette structure est spécifiée au moyen de l'attribut TYPE obligatoire. L'attribut TYPE peut faire référence à une structure ou à un élément de données EDD légal (par exemple, VARIABLE, ARRAY, RECORD, COLLECTION, LIST). L'accès aux éléments individuels de la LIST se fait à l'aide de la notation entre crochets, comme pour l'accès aux éléments ARRAY.

Un exemple d'utilisation d'une LIST dans un FILE est donné à la Figure 63. Dans cet exemple, la taille de la LIST (et donc du FILE) peut augmenter au fur et à mesure que sont stockées des formes d'onde radar significatives complémentaires qui documentent le fonctionnement du mesureur de niveau.

```
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 63 – Exemple de déclaration de fichier plus complexe

Dans l'exemple de la Figure 63, des informations de base ("gauge_location", "tag") concernant le mesureur de niveau sont stockées avec une série de signaux radar ("recorded_signals"). Dans cet exemple, "signal_info" est utilisé comme définition de type pour la liste. Le référencement de "signal_info" dans l'EDD n'accède pas à la liste. L'accès à la liste peut uniquement se faire à l'aide de la notation entre crochets. Les deux références suivantes se rapportent aux mêmes données.

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

Toutefois, les références suivantes n'accèdent pas aux mêmes informations.

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

La référence "echo_signal" définit la structure correspondant à une partie d'un élément de liste et, dans les deux cas, la quantité de données référencées est la même (dans les deux cas une matrice de 4 096 entiers non signés à 2 octets). Toutefois, la valeur "echo_signal" est nulle tant que des données n'y sont pas placées (en chargeant des données à partir de l'appareil de terrain, par exemple).

La Figure 64 représente un exemple de méthode utilisée pour l'examen des signaux radar stockés. Dans cet exemple, la construction LIST des formes d'onde stockées s'affiche de manière séquentielle. La variable "i" est utilisée pour exécuter pas à pas l'ensemble de la LIST, à l'instar d'un itérateur.

```
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
        }
}
/*
 ***************************************************************************
 * maintenant pour une méthode qui examine les signaux radar stockés (ping).
 *
 */
METHOD reviewStoredPings
{
        LABEL "SignalHistory";
        DEFINITION
        {
                int i,
                ans;            /*Les utilisateurs répondent pour sélectionner dans la liste*/
                long selection;
                long varid[5];      /*utilisé dans les appels Builtin de la méthode
acknowledge*/

                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
                {
                        /* obtenir l'enregistrement actuel afin de pouvoir l'afficher */
                        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;

                        /* afficher le graphique et les 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 64 – Exemple d'examen des signaux radar stockés

Cet exemple présente plusieurs fonctionnalités significatives. D'abord, le Builtin MenuDisplay permet l'utilisation des menus STYLE WINDOW ou DIALOG à utiliser au sein des méthodes.

Chaque itération appelle également la méthode "acknowledge()". Dans une méthode, l'objet graphique est séparé et affiché de manière asynchrone. Cela permet au développeur EDD d'utiliser les Builtins "acknowledge" (delay et put_message) pour interagir avec l'utilisateur exactement de la même manière que celle toujours utilisée dans les METHODs.

L'appel de la méthode "acknowledge" affiche l'opérateur et les commentaires concernant le signal enregistré. La date n'est pas affichée. Il est possible d'afficher la date après quelques ajustements, mais la mise en œuvre n'est pas représentée dans cet exemple.

Enfin, il convient de noter que le COUNT "recorded_signals.COUNT" (ainsi que les attributs FIRST, LAST et CAPACITY) est un attribut inhérent à toutes les constructions LIST. L'attribut COUNT peut être utilisé pour détecter la fin de l'élément LIST. L'accès à des éléments LIST non existants dans des METHODs entraîne l'interruption du processus.

Si l'attribut COUNT est spécifié dans une LIST, il définit sa taille lors de la création des données d'appareil d'instance. S'il n'est pas défini, la liste est vide.

Une liste dans une méthode peut être remplie en appelant le Builtin ListInsert ou en procédant à la lecture à l'aide d'une commande avec une variable d'information d'index. Les éléments d'une liste dans une méthode peuvent être supprimés en appelant ListDeleteElementAt.

L'index d'une liste commence à 0 et est toujours continu. Si un élément d'une liste est supprimé, les index des éléments suivants sont décrémentés. Si un élément est inséré, les index des éléments suivants sont incrémentés.

L'attribut COUNT peut être utilisé pour obtenir le nombre actuel d'éléments dans une liste. Il doit automatiquement être mis à jour avec ListInsert ou ListDeleteElementAt par l'application EDD.

La Figure 65 est plus impliquée. L'attribut procède à une mesure et relit le signal radar aux fins d'examen par l'opérateur. La commande 128 lance une mesure tandis que la commande 129 lit le signal renvoyé par l'appareil de terrain. Lorsque l'opération est terminée, le signal est affiché à l'utilisateur. L'utilisateur peut procéder à une autre lecture si celle-ci n'est pas satisfaisante.

Lorsqu'une lecture significative est trouvée, elle peut être ajoutée au FILE "stored_signals", remplacer un signal existant stocké dans "stored_signals" ou comparer la lecture actuelle à celle d'un signal enregistré. Dans de nombreux cas, les signaux stockés sont exécutés pas à pas comme dans l'exemple précédent.

Dans le cas d'une insertion, l'utilisateur peut insérer les nouvelles données au premier plan, au dernier plan ou au centre de la LIST contenue dans le FILE "stored_signals". Cette partie de code montre l'utilisation de l'attribut COUNT et de la fonction Builtin ListInsert().

La partie exécutant la fonction de remplacement ordonne la liste afin de trouver l'élément à remplacer. Un appel du Builtin ListDeleteElementAt() suivi d'un appel ListInsert().call affecte les remplacements.

Dans la dernière partie de l'exemple, le signal radar actuel est comparé à un signal stocké. La liste est exécutée pas à pas et le GRAPH "compare_radar_signals" affiche les deux signaux sur le même graphique.

```
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,     ping_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;
        }
}

/*
 ****************************************************************************
 * sonde le radar par pings et capture la forme d'onde de l'écho dans ping_data
 */
#define PING_COMPLETE 0x10;

METHOD newPing
{
        LABEL "Ping";
        DEFINITION
        {
         int i,
             ans;       /*Les utilisateurs répondent pour sélectionner dans la liste*/
                  long selector;
         long varid[5];     /*utilisé dans les appels Builtin de la méthode acknowledge*/

         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 );

         /*
          * un sondage par ping pour obtenir un signal radar
          */
         do {
             select_from_list ("Ready to make a measurement","Yes;No",ans);
             while (ans == 0) {
                 send(128);         /* ping */
                 do {             /* vérifier le statut du ping */
                     send(48);
                 } while(GET_VAR_VALUE (xmtr_specific_status4) & PING_COMPLETE);
             /*
              * ping terminé, lire le signal
              */
                 for (ping_index =0; ping_index < 4096; ping_index += 16) {
                     send (129);
                 }
                 MenuDisplay (new_radar_signal_menu, "OK", selector);
             /*
              * prendre pour hypothèse que le graphique est affiché jusqu'à ce qu'il soit
détruit
              */
                 select_from_list ("Take more radar data","Yes;No",ans);
             }

             /*
              * le signal radar a l'air correct. le sauvegarder?
              */
             select_from_list("what now?", "save the signal;" \
                                           "replace a stored signal; compare signals;" \
                                           "get new signal; quit", ans);
                       switch (ans)
                       {

                 case 0:        /* sauvegarder le 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;                      /* début */
                         case 1: i = recorded_signals.COUNT; break;    /* fin */

                         default:        /* insérer */
                             i = 0;
                             do {
                             /* obtenir l'enregistrement actuel afin de pouvoir l'afficher */
                                 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;
```

```
                /* afficher le graphique et les 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:         /* remplacer un signal stocké */
        i = 0;
        do {
        /* obtenir l'enregistrement actuel afin de pouvoir l'afficher */
            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;

        /* afficher le graphique et les 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:         /* comparer les signaux */
        i = 0;
        do {
        /* obtenir l'enregistrement actuel afin de pouvoir l'afficher */
            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;

        /* afficher le graphique et les 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:         /* obtenir un nouveau signal */
                    continue;

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

Figure 65 – Exemple d'EDD qui insère, remplace ou compare des signaux radar

5.2 Exposition des éléments de données en dehors de l'application EDD

Tous les éléments de données dont l'attribut VALIDITY est égal à FALSE ne doivent pas être exposés.

L'application EDD ne doit pas exposer les éléments de données en dehors de l'application EDD si l'attribut PRIVATE est évalué comme TRUE. Dans le cas d'un élément COLLECTION, RECORD ou REFERENCE_ARRAY où l'attribut PRIVATE est évalué comme FALSE ou n'est pas défini, l'application EDD doit exposer l'élément COLLECTION, RECORD ou REFERENCE_ARRAY, ainsi que l'ensemble des éléments référencés, quel que soit l'attribut PRIVATE des éléments de données référencés.

Dans le cas d'un élément COLLECTION, RECORD ou REFERENCE_ARRAY où l'attribut PRIVATE est évalué comme TRUE, l'élément COLLECTION, RECORD ou REFERENCE_ARRAY n'est pas exposé, mais les éléments de données référencés peuvent être exposés au moyen d'autres règles.

5.3 Initialisation d'instances EDD

5.3.1 Vue d'ensemble

L'initialisation de variables est importante pour la configuration hors ligne. Si un appareil est configuré, par exemple lors de la phase de planification, la configuration peut être réalisée en cohérence avec les appareils. Le développeur EDD définit des plages et des énumérations avec des expressions conditionnelles pour permettre à l'application EDD de vérifier la cohérence des données et d'éviter la survenue de problèmes pendant le téléchargement descendant de la configuration.

5.3.2 Prise en charge de l'initialisation

Si l'utilisateur choisit un appareil pour la première fois, une nouvelle instance est créée par l'application EDD avec l'EDD associée. Les valeurs des variables EDD pour les instances nouvellement créées doivent être initialisées à l'aide des règles suivantes:

a) initialisation de Variable, Value Array et List en utilisant les COMPONENTs EDDL s'ils existent; sinon

b) initialisation de Variable avec des INITIAL_VALUEs EDDL si elles existent; sinon

c) initialisation de Variable avec des DEFAULT_VALUEs si elles existent; sinon

d) variables initialisées avec la valeur initiale de type variable.

L'utilisateur peut en outre choisir un TEMPLATE avec d'autres initialisations.

5.3.3 TEMPLATE

Les TEMPLATEs spécifient des valeurs de paramètre par défaut pour différentes utilisations d'un appareil; autrement dit, ils sont spécifiques à l'application. L'application EDD doit énumérer tous les TEMPLATEs définis dans l'EDD pour l'utilisateur. Si l'utilisateur sélectionne un TEMPLATE, l'application EDD doit réinitialiser toutes les VARIABLEs spécifiées dans le TEMPLATE avec la valeur définie. L'application EDD doit autoriser l'utilisateur à appliquer le même TEMPLATE ou un autre plusieurs fois; la dernière modification de la valeur est conservée.

L'initialisation du TEMPLATE de la VARIABLE employée dans l'attribut TYPE d'une ARRAY ou d'une LIST ne doit pas affecter les éléments de l'ARRAY ou de la LIST.

5.4 Mapping de modèle d'appareil

5.4.1 BLOCK_A

Une EDD FOUNDATION fieldbus ou ISA100_Wireless peut contenir des menus racine au niveau de l'appareil (par exemple, process_variables_root_menu) ou des menus de niveau de bloc (par exemple, process_variables_root_menu_ai).

Tous les MENU ITEMS spécifiés dans un menu au niveau de l'appareil doivent explicitement spécifier le bloc avec lequel ils sont associés.

Les menus de niveau de bloc doivent être référencés dans l'attribut MENU_ITEMS BLOCK_A. Les menus de niveau de bloc ne doivent pas inclure des éléments de menu qui font référence à un bloc spécifique. Par exemple, cela signifie qu'un menu ou une méthode avec une VALIDITY conditionnelle qui elle-même contient une référence à un bloc spécifique peut ne pas se trouver dans un menu de niveau de bloc, directement ou indirectement.

L'exemple de la Figure 66 représente le référencement, conformément au bon contexte.

```
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_variables_root_menu_ai;   /* menu de bloc */
   }
}

MENU process_variables_root_menu  /* menu au niveau de l'appareil */
{
   ITEMS
   {
      __analog_input_block[0].PARAM.ST_REV;/* référence de bloc spécifiant l'instance de bloc
*/
   }
}

MENU process_variables_root_menu_ai /* un menu de niveau de bloc */
{
   ITEMS
   {
      PARAM.ST_REV;    /* le menu est associé à un type de bloc, */
                       /* seul le référencement des paramètres est ainsi utilisé */
   }
}
```

Figure 66 – Exemple de BLOCK_A

5.4.2 BLOCK_B

Les EDD pour les appareils PROFIBUS et PROFINET doivent contenir tous les menus nécessaires au niveau de l'appareil. Les menus de niveau BLOCK_B n'existent pas pour les appareils PROFIBUS et PROFINET.

La définition BLOCK_B n'est utilisée que pour l'adressage des données d'appareil des appareils de contrôle des processus pour les profils PI.

6 Programmation avec la METHOD EDDL et utilisation de builtins

6.1 Environnement de la méthode

6.1.1 Généralités

L'ensemble des conventions lexicales spécifiées dans l'IEC 61804-3 s'applique aux méthodes et doit être utilisé dans le cadre du développement de la DEFINITION fondée sur le langage ANSI C de la procédure spécifiée par la METHOD. La traduction a été réalisée à partir de la source EDD en tenant compte de l'exécution des commandes du préprocesseur avant le moment où l'application hôte a accès à l'EDD ou l'utilise.

6.1.2 Sécurité

Les méthodes sont exécutées dans un "bac à sable" sécurisé inclus dans l'application EDD. Toutes les activités liées à la méthode se produisent dans ce bac à sable. Les applications EDD doivent exclusivement déployer les fonctions normalisées de bibliothèque de BuiltIn. Ces fonctions de bibliothèque ne peuvent pas lancer des programmes issus de l'application EDD ou avoir accès à une bibliothèque de liens dynamiques (DLL, *Dynamic Link Library*) externe. L'accès direct au disque local du système ne doit pas être autorisé. Le domaine d'application concentré et limité des fonctions normalisées de la bibliothèque assure la sécurité de l'utilisation des EDD et contrôle l'accès aux ressources du système depuis l'intérieur des méthodes EDD. L'application EDD est responsable de sa propre sécurité et de la sécurité du système sur lequel elle opère.

D'un point de vue conceptuel, ce bac à sable est créé lorsqu'une méthode est appelée ou déclenchée. Le bac à sable existe jusqu'à ce que la méthode se termine normalement ou par l'interruption de la méthode. L'ensemble des sous-programmes de la méthode et des méthodes d'interruption est exécuté dans le bac à sable créé au début de la méthode.

6.1.3 Données de l'appareil

Toutes les données d'appareil telles que les VARIABLES et ARRAYS apparaissent en tant que variables globales du code C qui définit la procédure. L'accès aux VARIABLES peut être direct, comme pour toute variable déclarée dans le langage ANSI C.

Normalement, le cache de données primaire de l'appareil est mis à jour de manière continue par l'application EDD afin de garantir qu'elle dispose d'une copie à jour des données contenue dans l'appareil de terrain. Lorsque le bac à sable est créé, le cache de données (secondaire) de la méthode est initialisé à partir du cache primaire. Si le cache primaire n'est pas totalement initialisé, les commandes nécessaires sont envoyées pour les éléments de données référencés dans la méthode. A des fins de sécurité, la méthode opère sur cet ensemble de valeurs mis en cache plutôt que de manipuler directement le cache primaire. Cela permet, par exemple, d'interrompre une méthode sans que cela ne compromette la configuration de l'appareil de terrain.