BS ISO 20242-5:2020



BSI Standards Publication

Industrial automation systems and integration — Service interface for testing applications

Part 5: Application program service interface

bsi.

AVA Documents
Document Sharing Hub

Full document details: BS ISO 20242-5:2020

National foreword

This British Standard is the UK implementation of ISO 20242-5:2020.

The UK participation in its preparation was entrusted to Technical Committee AMT/5, Industrial architectures and integration frameworks.

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

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

© The British Standards Institution 2020
Published by BSI Standards Limited 2020

ISBN 978 0 580 91600 7

ICS 25.040.40

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

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 30 June 2020.

Amendments/corrigenda issued since publication

Date	Text affected

Documents
Document Sharing Hub

Full document details: BS ISO 20242-5:2020

INTERNATIONAL STANDARD

ISO 20242-5

First edition
2020-06

Industrial automation systems and integration — Service interface for testing applications —

Part 5: Application program service interface

Systèmes d'automatisation industrielle et intégration — Interface de service pour contrôler les applications —

Partie 5: Interface de service des programmes d'application

Reference number
ISO 20242-5:2020(E)

Documents
Document Sharing Hub
© ISO 2020

Full document details: BS ISO 20242-5:2020



COPYRIGHT PROTECTED DOCUMENT

© ISO 2020

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

© ISO 2020 – All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO 20242-5:2020

Contents

Page

Foreword iv
Introduction v
1 Scope 1
2 Normative references 1
3 Terms and definitions 1
4 Symbols and abbreviated terms 3
5 Application Program Service Interface 3
5.1 Introduction 3
5.2 Parameterization 5
5.3 Configuration 6
5.3.1 PID based configuration 6
5.3.2 Service based and mixed configuration 6
5.4 Coordinator structure 7
5.4.1 Workspace 7
5.4.2 Interface variants 9
5.5 Details for using APSI 12
5.5.1 Workspace structure 12
5.5.2 Callback methods and references 12
5.5.3 Workspace extension 13
5.5.4 Workspace transfer 14
5.5.5 Data monitoring 14
5.5.6 Data types 15
5.5.7 Streaming 16
5.6 Error handling 18
Annex A (normative) Programmer's reference guide — Smart Access Interface 19
Annex B (normative) Programmer's reference guide — Extended Access Interface 79
Annex C (normative) Programmer's reference guide — Full Access Interface 125
Annex D (normative) Reference guide — Enums and status/ident informations 222
Annex E (informative) Sequence diagrams 235
Annex F (informative) Streaming 243
Annex G (informative) Data alignment and byte order 247
Annex H (informative) Testing applications with the ISO 20242 (APSI) series and the ISO 13209 (OTX) series 250
Annex I (normative) Conformance test methods, criteria and reports 279
Bibliography 283

© ISO 2020 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO 20242-5:2020

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/TC 184, *Automation systems and integration*, Subcommittee SC 5, *Interoprability, integration and architectures for enterprise systems and automation applications*.

A list of all parts in the ISO 20242 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

© ISO 2020 – All rights reserved
Documents Document Sharing Hub

Full document details: BS ISO 20242-5:2020

Introduction

The motivation for the ISO 20242 series stems from international automotive industries and their suppliers to facilitate the integration of automation and measurement devices, and other peripheral components for this purpose, into computer based applications. It defines rules for the construction of device drivers and their behaviour in the context of an automation and/or measurement application.

The main goal of the ISO 20242 series is to provide users with:

— independence from the computer operating system;

— independence from the device connection technology (device interface/network);

— independence from device suppliers;

— the ability to certify device drivers with connected devices and their behavior in the context of a given computer platform;

— independence from the technological device development in the future.

The ISO 20242 series will not force the development of new device families or the use of special interface technologies (networks). It encapsulates a device and its communication interface to make it compatible with other devices of that kind for a given application.

© ISO 2020 – All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO 20242-5:2020

Documents
Document Sharing Hub

Full document details: BS ISO 20242-5:2020

Industrial automation systems and integration — Service interface for testing applications —

Part 5:
Application program service interface

1 Scope

This document defines the formatting, syntax and semantic rules for describing

— an object oriented interface for using services provided by a coordinator and

— the configuration of virtual devices and the environment for their use.

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.

ISO 20242-1, *Industrial automation and systems integration — Service interface for testing applications — Part 1: Overview*

ISO 20242-2, *Industrial automation and systems integration — Service interface for testing applications — Part 2: Resource Management Service Interface*

ISO 20242-3, *Industrial automation and systems integration — Service interface for testing applications — Part 3: Virtual Device Service Interface*

ISO 20242-4, *Industrial automation and systems integration — Service interface for testing applications — Part 4: Device Capability Profile Template*

ISO 13209-1, *Road vehicles — Open Test sequence eXchange format (OTX) — Part 1: General information and use cases*

ISO 13209-2, *Road vehicles — Open Test sequence eXchange format (OTX) — Part 2: Core data model specification and requirements*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO 20242-1, ISO 20242-2, ISO 20242-3, ISO 20242-4, ISO 13209-1, ISO 13209-2 and the following apply.

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

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at http://www.electropedia.org/

© ISO 2020 – All rights reserved

Documents Document Sharing Hub

Full document details: BS ISO 20242-5:2020

3.1
communication object
existing object which may be accessed with a communication function to read or write a value

[SOURCE: ISO 20242-1:2005, 2.3]

3.2
coordinator
program with a specified interface to handle the access of an application program to one or more *device drivers* (3.5) and to manage real-time application aspects, synchronization and events

[SOURCE: ISO 20242-1:2005, 2.4]

3.3
coordinator capability description
text file containing information about the capabilities of *coordinators* (3.2) in a defined format (i.e. structure, syntax)

[SOURCE: ISO 20242-4:2011, 3.3]

3.4
coordinator services
services of a *coordinator* (3.2) for the exchange of data with application programs

3.4
device capability description
text file containing information about the capabilities of *virtual devices* (3.9) in a defined format (i.e. structure, syntax)

[SOURCE: ISO 20242-1:2005, 2.5, modified — Note 1 to entry deleted.]

3.5
device driver
software module providing an ISO 20242-specified interface with service functions to call a platform adapter to access physical devices

[SOURCE: ISO 20242-2:2010, 3.1]

3.6
function object
instance describing one capability of a *virtual device* (3.9)

[SOURCE: ISO 20242-3:2011, 3.4]

3.7
operation
instance describing one complete procedure

[SOURCE: ISO 20242-3:2011, 3.5]

3.8
parameterization instance description
information about the configuration of a *coordinator* (3.2) and of *virtual devices* (3.9)

[SOURCE: ISO 20242-4:2011, 3.8]

© ISO 2020 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO 20242-5:2020

3.9
virtual device
representation of one or more physical devices and/or stand-alone software modules that provide an unambiguous view of the resources of a communication interface

[SOURCE: ISO 20242-3:2011, 3.7]

3.10
workspace
grouping of *coordinators* (3.2) resources providing an access point for an application

4 Symbols and abbreviated terms

APSI	Application Program Service Interface
ASCII	American Standard Code for Information Interchange
CCD	Coordinator Capability Description
CO	Communication Object
DCD	Device Capability Description
DCPT	Device Capability Profile Template
DSL	Domain Specific Language
EAI	Extended Access Interface
FAI	Full Access Interface
FO	Function Object
NIL	Null pointer or null reference, does not refer to a valid object
OP	Operation
OTX	Open Test sequence eXchange
PID	Parameterization Instance Description
SAI	Smart Access Interface
VD	Virtual Device
VDSI	Virtual Device Service Interface
XML	eXtensible Markup Language (see REC-xml-20040204)

5 Application Program Service Interface

5.1 Introduction

This clause describes a set of services that shall be provided by a coordinator to be used by several application programs (Figure 1).

© ISO 2020 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO 20242-5:2020

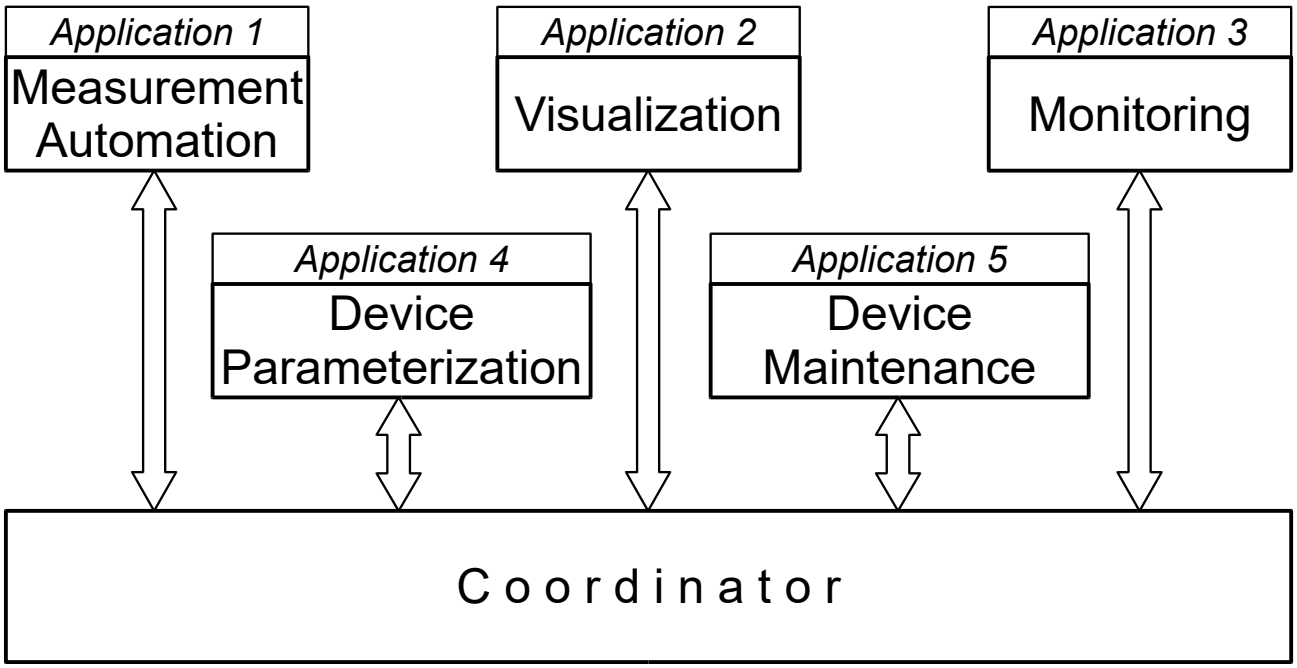


Figure 1 — Exemplary application programs using a coordinator

The concept for the Application Program Service Interface (APSI) is to be device neutral. It shall be possible to access application-relevant data presented by any device without device-specific services or parameters (Applications 1, 2 and 3 in Figure 1). Nevertheless, for the configuration of devices and other device-specific tasks it should also be possible to transfer device-specific information (Applications 4 and 5 in Figure 1). Therefore, APSI defines different interfaces (see 5.4.2), whereas the Smart Access Interface (SAI) is mandatory and shall not handle any device-specific access, and the Full Access Interface (FAI) is optional and shall enable an access to virtual devices in device drivers (see ISO 20242-3). In a typical full stack usage of the ISO 20242 series, the coordinator accesses device drivers via the Virtual Device Service Interface (see Annex I and ISO 20142-1:2005).

The application should use abstract data sources and sinks identified by instance names. Depending on the application environment, data types of sources and sinks are known before or have to be identified via the interface. Applications using APSI typically will not have information about the physical devices providing the requested information processing. In addition, the coordinator shall implement a memory separation between application data and device data (see 5.5.7). Therefore, device-specific data elements shall be encapsulated and not be visible for the application.

This document describes the basics of coordinator services and their usage in typical applications. An object-oriented model is used with a fundamental description of classes, attributes and methods. Services are executable synchronous by default. The asynchronous operation is optional. Corresponding capabilities of a coordinator shall be described in a datasheet delivered with the coordinator (see I.2). Asynchronous operation is controlled via callback functions. For global event handling in a workspace (see 5.4.1), the callback functions (e.g. see `refCBInformationReport` in A.2.2.7) shall be defined with opening the workspace. Additionally local callbacks should be registered at each attribute (see 5.5.2). Coordinator services will typically be mapped to the applied technology, platform and programming language (e.g. CORBA, JAVA, C++). Therefore, this document defines services and not an interface for a specific programming language. Details for a specific programming language shall be explained for the respective mapping in a datasheet (see I.2). This also applies for the error handling, because technological restrictions of data processing have to be considered for the corresponding implementation.

Annex H describes the usage of this standard with applications based on ISO 13209 (OTX). Conformance tests for this document shall be carried out as specified in Annex I.

© ISO 2020 – All rights reserved
AVA Documents Document Sharing Hub

Full document details: BS ISO 20242-5:2020

5.2 Parameterization

The coordinator shall be able to link and release applications and device drivers dynamically and to create and delete data objects in device drivers (virtual devices, function objects and communication objects as defined in ISO 20242-3). This procedure shall be identical for all device drivers. The sequence of creating and deleting data objects shall not be fixed. It shall be possible to use a parameter instance description (PID, as defined in ISO 20242-4) to control the procedure of instantiation.

NOTE 1 The coordinator does not know, which device capabilities are necessary for a specific application, but it can communicate with each device on the basis of its device capability description (DCD, as defined in ISO 20242-4).

NOTE 2 To achieve a balance between the functionalities needed by the application and the device capabilities provided by the device drivers, a configuration is usually set up by means of a parameterization procedure. The result is a parameterization instance description (PID) which contains the information for the coordinator to create all necessary data objects. This procedure is the key for device independence, as the application does not need to know the specific capabilities of the used devices. A parameterization tool can create the PID, which can be part of the application itself or a stand-alone software.

The parameterization shall be based on the DCD available for the device drivers and should be supported by multilingual text elements as defined in ISO 20242-4:2011, Clause 8.

NOTE 3 The PID can be created without a coordinator and it is not necessary to connect any device.

The PID shall not be altered by a coordinator.

The following information shall be included within the parameter instance description:

— workspace name,

— drivers (location) with version number,

— device capability descriptions (DCD) of drivers,

— virtual devices (VD) including identification and parameters for creation,

— function objects (FO) including identification and parameters for creation,

— communication objects (CO) and their default values,

— operations (OP) and their identification,

— in/out-structures of operations with values for input parameter,

— initializing order, and

— application references (instance names) for the abstract data sources and sinks.

The parameter instance description (PID) shall be based on the description of structured data types. Thus, the typedef sections of the DCDs shall be resolved. See ISO 20242-4 for details.

NOTE 4 With this parameterization, the initialized values for virtual devices, function objects and communication objects are established for an application-related configuration.

The generated PID shall be stored separately. The read configurations shall be reproducible by the coordinator at any time.

© ISO 2020 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO 20242-5:2020

5.3 Configuration

5.3.1 PID based configuration

With a PID based configuration, an XML file created using the specification in ISO 20242-4 will be presented to the coordinator by the application at the creation of a workspace. This file shall be processed automatically by the coordinator. When an application logs in at the coordinator, the instance description to be used shall be identified by name.

If the application knows the instance names and types of the abstract data sources and sinks, it could iterate over the object references created by the coordinator. This is a typical use case for the Smart Access Interface.

If the application does not know the instance names and types of the abstract data sources and sinks, it should iterate over the object references created by the coordinator and request the names and types of the abstract data sources and sinks. This is a typical use case for the Extended Access Interface.

The coordinator shall instantiate the objects given with the instance description, and thus creates abstract data sources and inside to be used by the application. The data objects of the device driver assigned to these abstract data sources and sinks shall be instantiated within the corresponding virtual devices.

NOTE During the life time of the device driver these abstract data sources and sinks can be updated by the application via reading and writing. The coordinator can update the abstract data sources and sinks accessing the data objects instantiated within the device drivers, as the coordinator knows the relations between the abstract data sources and sinks inside and the function objects of the virtual devices.

The operating state of a virtual device will be "Working" (ISO 20242-3:2011, Clause 7) after running a parameterization with the SAI or EAI. In this state, the modification of parameters is not possible. The method `GdiCoWorkspace::gdiSetAllowChangeParameter(boolean bEnable)` resets this prohibition and enables the application to write a parameter (see A.2.29.12). For this, the coordinator shall transit the respective VDs to the appropriate states.

5.3.2 Service based and mixed configuration

If the full access interface is implemented in the coordinator, an application itself can do the configuration without an accordingly prepared PID.

In a service based configuration, the structure (classes and data types) and the content (instances and parameters) of the interface can be created. See Annex C for details of this step by step procedure of configuration.

In a mixed configuration, a PID without instances and parameters (called empty PID in this document) can be presented at the interface. The coordinator shall build the structure based on the DCD contained in the empty PID. Instances and parameters can be created afterwards.

For both cases, the operating state of a VD can be controlled by the application.

© ISO 2020 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO 20242-5:2020

5.4 Coordinator structure

5.4.1 Workspace

The workspace in this document is defined as a grouping of coordinator resources and provides an access point for an application. The coordinator shall provide several workspaces which shall be managed separately. An application can use one or more workspaces. Workspaces shall be identified by name and shall connect only to one application (Figure 2) and an optional monitoring application (see 5.5.5 for details).

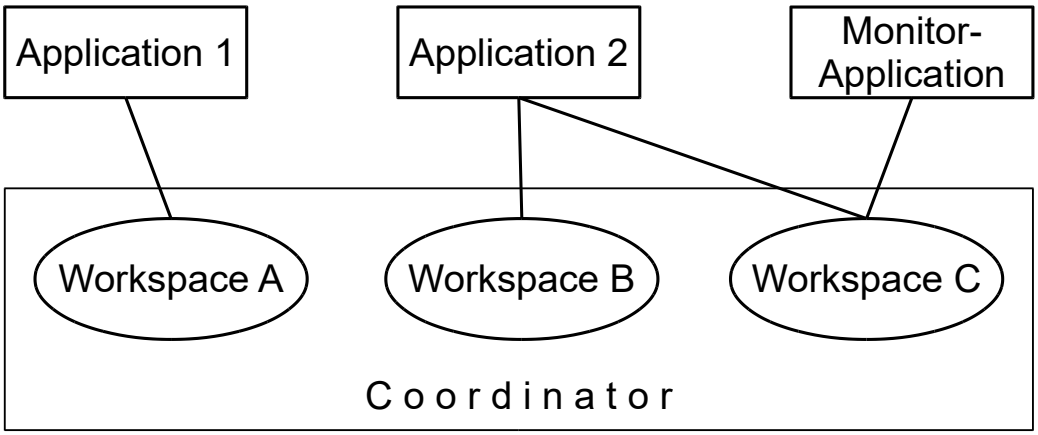


Figure 2 — Applications and workspaces

APSI shall be a multi client interface, as several workspaces can be used at the same time. The content of a PID file shall exactly describe one workspace, independent from the number of devices, device drivers and DCDs.

© ISO 2020 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO 20242-5:2020

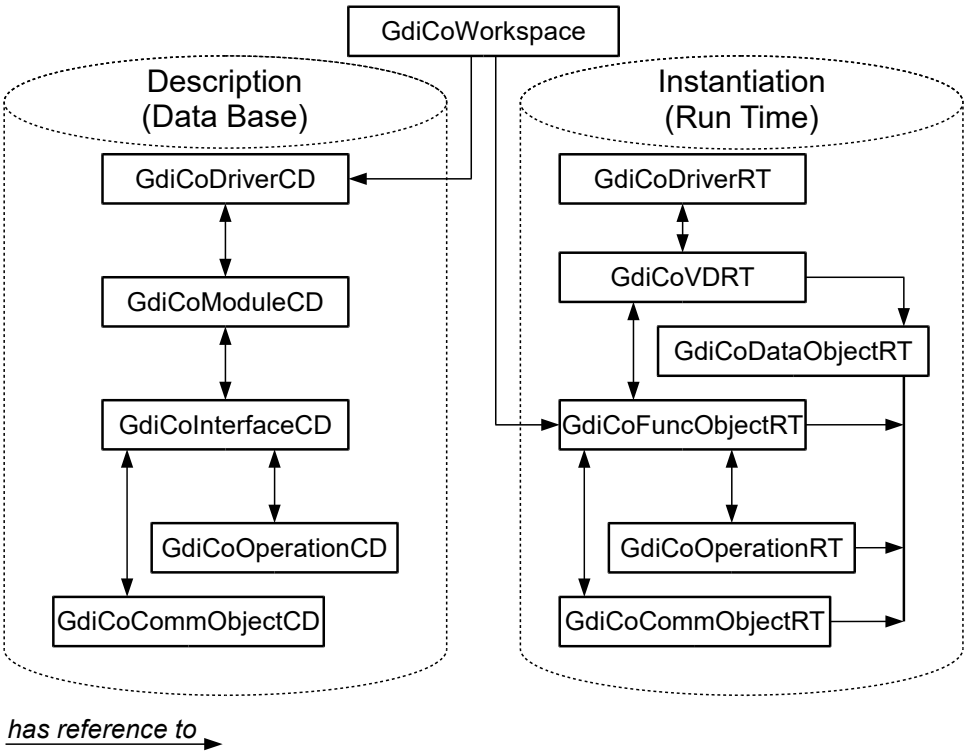


Figure 3 — Workspace view for service organisation

With connecting to a workspace, an application shall get a handle for its further identification. Therefore any interface access after connecting takes this application handle as parameter.

A coordinator shall be able to handle several applications simultaneously. The objects set up by the coordinator for each application shall remain reserved for this application and shall be invisible for other applications. If several applications need access to identical coordinator objects, this should be organized by the application connected to the workspace.

Access services defined in this document shall be carried out as specified in Annex A for the Smart Access Interface (SAI), Annex B for the Extended Access Interface (EAI) and Annex C for the Full Acces Interface (FAI). Further services providing information about enums, status and identification shall be carried out as specified in Annex D. The services are organized with a view on two sides of a workspace. One side is called Description and the other side is called Instantiation (see Figure 3).

The Description side is also referred to as Data Base side, because the contained information is static and represents the class descriptions from the DCDs of the corresponding device drivers. The Instantiation side is referred to as RunTime side, because it represents the realized instances including default initial values used at RunTime. If an application uses only the services from the RunTime side, the allocation of the separate device drivers and VDs shall be not visible. Instead a pool of abstract data sources and sinks shall be provided.

Device driver and DCD form a unit. For each device driver there is exactly one Description side, which represents the respective DCD, and exactly one RunTime side. The classes of Description side and the objects of RunTime have a tree structure. Each element is assigned to an unambiguous position in the tree. The `GdiCoDriverCD` (see C.2.7) is the root element in the Description side and `GdiCoDriverRT` (see C.2.9) is the root element in the RunTime side.

© ISO 2020 – All rights reserved
Documents
Document Sharing Hub
Full document details: BS ISO 20242-5:2020

Instance names of function objects within a workspace (RunTime side) shall be unique over all used drivers.

If an object from RunTime side is set up, there shall be an implicit set up of the corresponding instance within the device driver. All parameters for set up shall be handed over directly by streaming (see Annex F) when the service for the creation of a VD or FO is used.

Annex E describes typical use cases to get more grip on the services and to facilitate the development of coordinator software.

5.4.2 Interface variants

This document defines three kinds of interfaces.

For PID based configuration there shall be the

— mandatory Smart Access Interface

and there should be the

— optional Extended Access Interface.

For service based or mixed parameterization there should be the

— optional Full Access Interface.

With view to the structure of APSI, these different interfaces are based upon each other. There is additional functionality within the Extended Access Interface compared to the Smart Access Interface. The classes and methods of individual objects are always identical for each interface variant (e.g. `GdiCoFuncObjectRT`). An application can use any of the interface variants, if they all are implemented. With creating a workspace, the parameter `eRequiredCoordinatorInterface` shall select the required functionality.

With the Extended Access Interface, only the object `GdiCoExtendedObjectNavigator` (see B.2.6, it contains methods for navigation over the objects and data) is defined additional to the Smart Access Interface. With the Full Access Interface the classes `GdiCoObjectNavigator` (see C.2.18, it contains methods for navigation over the objects and data) and `GdiCoFactory` (see C.2.11, it contains methods for adding and deleting of objects) exist in addition to the Extended Access Interface. The Full Access Interfaces shall be able to alter (extend, modify, delete) the RunTime side (Instantiation) and to also alter (extend, modify) the Data Base side (Description). The capabilities of the different interfaces shall be independent from the kind of configuration (PID based, service based or mixed) done after creating the workspace.

To maintain the compatibility between the separate interfaces, the methods

— `gdiGetFactory`

— `gdiGetObjectNavigator` and

— `gdiGetExtendedObjectNavigator`

are included in the class `GdiCoWorkspace` (see A.2.29). They return the references to the objects corresponding to the capabilities of the coordinator, or NIL, if the Coordinator does not support the desired functionality.

© ISO 2020 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO 20242-5:2020

With the description of interfaces in this document, the stereotypes explained in Table 1 are used to mark the affiliation of classes and methods to an interface variant.

Table 1 — Classes of interface variants (without data types)

Stereotype	Applies to
<< S, E, F >>	classes and methods of Smart Access Interface.
	Also available in Extended Access Interface and Full Access Interface.
<< E, F >>	classes and methods of Extended Access Interface.
	Also available in Full Access Interface.
<< F >>	classes and methods of Full Access Interface.
	Only available in Full Access Interface.

In figures, stereotypes are marked with graphical elements (Figure 4).

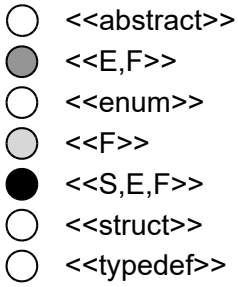


Figure 4 — Graphical marks for stereotypes in figures

Figure 5 shows the classes for all interface variants. If implemented, the coordinator shall provide the interface variant requested by an application. A workspace created for Smart Access Interface shall be open for switching to Full Access Interface, if that is implemented. After using a workspace for Smart Access Interface with configuration by PID, it shall be possible to transfer the workspace to another application for being used with its Full Access Interface to extend the capabilities by additional service based configuration. See 5.5.4 for details.

The method `GdiCoManager::gdiGetCoordinatorType` shall return a bit mask, which visualizes the support of the interface variants (<S>, <E>, <F>).

© ISO 2020 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO 20242-5:2020

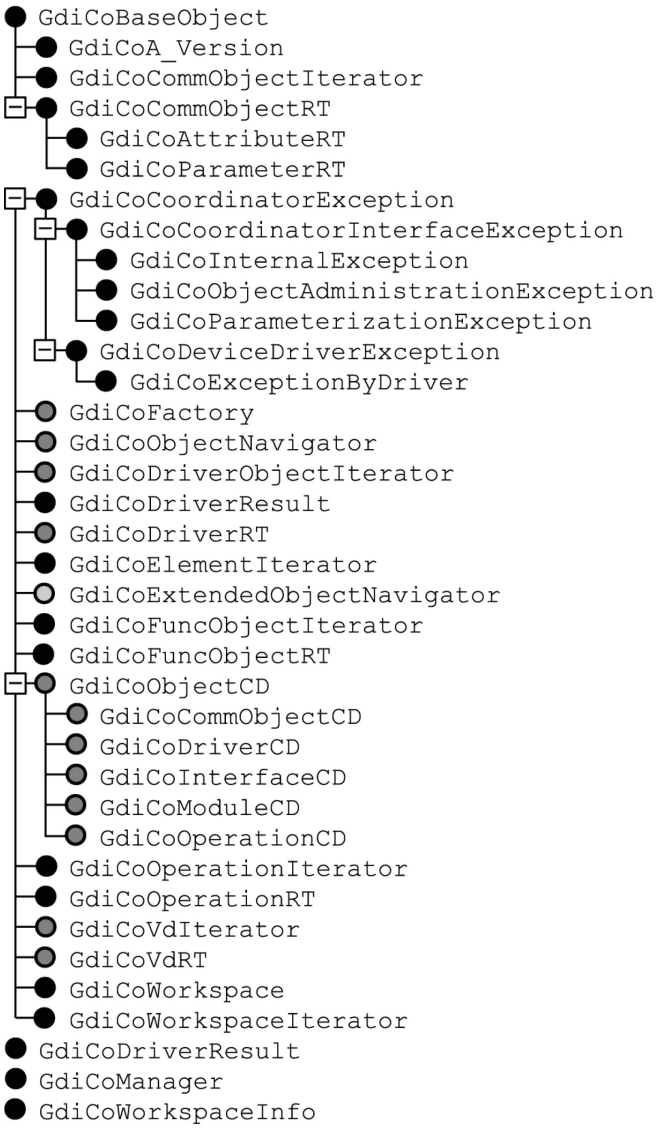


Figure 5 — Classes of interface variants (without data types)

The Application Programm Service Interface with implemented Full Access Interface shall support all services described in Annex C and therefore provides capabilities to

— set up application areas within the coordinator and to generate instances via parameterization data or via services,

— obtain references to function objects and communication objects,

— exchange data with the coordinator,

— initiate the data transport within a device driver,

— obtain descriptions of data (types and names),

— load device drivers,

— install and control virtual devices,

© ISO 2020 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO 20242-5:2020

— generate instances (function objects and communication objects),

— set up data elements (assign symbolic names and define types) and

— delete instances.

5.5 Details for using APSI

5.5.1 Workspace structure

For each device driver and DCD exactly one Description side and one Instantiation side exists in the workspace. These are structured as trees. The application access to the individual objects depends on the used interface variant. See Figure 5 for an overview of the classes and objects which can be used by an application. The names of the classes in Description side end with CD (for capability description) to show their affiliation as the names in RunTime side end with RT.

RunTime objects can be instantiated using the reference to the Description objects, the class name (e.g. name of an interface in the DCD) or the class Id (DCD Id). A coordinator shall provide the following services for the instantiation of RunTime objects:

`gdiCreateDriverRTByDescription()`,
`gdiCreateVdRTByDescription()`,
`gdiCreateFuncObjectRTByDescription()`,
`gdiCreateCommObjectByDescription()` and
`gdiCreateOperationRTByDescription()` in `GdiCoFactory` (see C.2.11).

From RunTime side, the respective Description objects may be accessed via methods of the RunTime objects. That is:

`gdiGetDriverCD()` in `GdiCoDriverRT` (see C.2.9),
`gdiGetModuleCD()` in `GdiCoVdRT` (see C.2.29),
`gdiGetInterfaceCD()` in `GdiCoCommObjectCD` (see C.2.4),
`gdiGetCommOjectCD()` in `GdiCoObjectNavigator` (see C.2.18) and
`gdiGetOperationCD()` in `GdiCoObjectNavigator` (see C.2.18).

If the Full Access Interface is used, the Description elements as well as the RunTime objects may be modified.

5.5.2 Callback methods and references

An asynchronous execution of services should be implemented, as it provides an easy way for applications to handle concurrent procedures for data transfer.

If asynchronous execution is implemented, it shall be realized via callback methods.

With creating a workspace by method `gdiCreateWorkspace()` in `GdiCoManager` (see A.2.20), the arguments `refCBException`, `refCBAccept` und `refCBInformationReport` shall take callback references to methods within the application for handling event based errors, unsolicited data requests by devices and unsolicited data transmissions from devices. With a NIL reference, the callbacks shall be switched off.

This global assignment of callback references shall be mandatory for the whole life cycle of the workspace. It shall be possible to overwrite the callbacks locally.

© ISO 2020 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO 20242-5:2020

The local callback registration shall be done with the

— create methods (VD, FuncObject, ComObject), and the

— asynchronous method calls (`GdiCoCommObjectRT::gdiReadValue`, `GdiCoCommObjectRT::gdiWriteValue`, `GdiCoOperationRT::gdiExecuteOperation`, `GdiCoExtendedObjectNavigator::gdiReadCommObjectData`, `GdiCoExtendedObjectNavigator::gdiWriteCommObjectData` and `GdiCoExtendedObjectNavigator::gdiExecuteOperation`).

The local callback reference is handed over with this registration and temporarily overwrites the global callback reference for the respective application (identified by its handle). If more applications are registered (see data monitoring 5.5.5), the callbacks shall be distributed to all registered applications. For deregister, the callback reference NIL shall be given. It shall be possible to change local references for InformationReport and Accept by multiple registrations.

With `gdiReleaseWorkspace` or `gdiReleaseMonitorAccess` the registered callbacks shall be made invalid. If an application connects to an existing workspace, the callback references shall be registered again.

The same applies for InformationReport and Accept. The global callback references shall be set at creating the workspace and the local callback references at the `GdiCoAttributeRT` (`gdiSetLocalCBAccept`, `gdiSetLocalCBInfReport`).

5.5.3 Workspace extension

It shall be possible to extend existing configurations by using a (new) PID file. This shall not depend on the used interface variant.

For this, the following rules apply:

— The existing configuration shall be maintained;

— No information shall be deleted;

— Configuration elements shall only be added;

— The PID file has to belong to the same devices.

At first the correct syntax and semantic of the PID file shall be checked with respect to the definitions of ISO 20242-4.

For the following configuration check, the PID shall be aligned with the existing configuration of the workspace. The new PID file shall be tested for already existing objects. Instance names and the DCD class descriptions for objects (OP, CO, FO) shall be compared. For already existing objects, the application reference name and the DCD class name for the object and the parent (FO and/or VD) shall be in accordance. If there is any CO or OP with an application reference name that has not been instantiated and the application reference name and DCD class name for the parent (FO/VD) are identical, a re-instantiation shall be done. Any creation parameters for already existing VDs and FOs shall be ignored.

If application reference names for identical class names of CO and/or OP are not in accordance, an error exception shall be thrown (see 5.6). If application reference names for identical class names of FOs are not in accordance but the parent object has an identical application reference name and DCD class name for the VD, this FO shall be added.

© ISO 2020 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO 20242-5:2020

If the new PID file holds values for existing COs, these COs shall be written for a value update. Create parameters of existing FOs generally shall not be updated.

All other COs shall not be modified. Also, no implicit default values shall be set (e.g. minimum of a value).

If the new PID file marks execution of operations, they shall be executed.

When method `gdiExtendWorkspace` is finished, the state of the corresponding VD shall be Working (see ISO 20242-4).

If any error concerning accordance is registered for the given PID file, the processing is stopped. The Workspace remains in the VD state Preparation for all VDs concerned. The same action will take place, if any error occurs while running the PID file.

If there is an error, the processing is stopped. All concerned VDs shall be in state Preparation (see ISO 20242-4). If the error is with the syntax and semantic check, the method `gdiExtendWorkspace` shall throw the exception

`eINT_PID_FILE_MISMATCH` (GdiCoInternalException, see A.2.19)

and if it is with applying the new configuration, the method shall throw the exception

`ePAR_INCORRECT_PARAMETERIZATION` (GdiCoParameterizationException, see A.2.24).

5.5.4 Workspace transfer

It shall be possible that different applications can use the same workspace successively. When an application disconnects from the workspace via method `gdiReleaseWorkspace` (see A.2.20.11), another application may connect to the released workspace using the workspace name and method `gdiGetWorkspaceByName` (see A.2.20.7).

NOTE The workspace transfer enables the user of APSI to split big applications into smaller, independent modules which take less resources. Examples are modules for basic configuration, extend configuration, maintain devices and run automation.

Because of the sequential use of the workspace by different applications, there is always a short period of time, where no application is connected. When executing `gdiReleaseWorkspace()`, the coordinator shall set all connected VDs to the state Preparation, and release the workspace afterwards. Release of workspace shall be rejected, if errors hinder the transition to Preparation.

With `gdiGetWorkspaceByName()` the VDs shall be set to Working if SAI or EAI is used (`eSMART` or `eEXTENDED in parameter eRequiredCoordinatorInterface`) or shall remain in Preparation if FAI is used (`eFULL i`n the parameter `eRequiredCoordinatorInterface`).

The state of a workspace, e.g. if it is just used or not, shall be provided for any application via GdiWorkspaceIterator (see A.2.31) and GdiCoWorkspaceInfo (see A.2.30).

After release of a workspace, all references shall become invalid. Any access with an invalid handle shall throw the exception `eINT_INVALID_ACCESS` (see A.2.19).

5.5.5 Data monitoring

If an application connects to a workspace via `gdiGetMonitorAccessByWorkspaceName` (see A.2.20.6), it shall have only rights for reading data access (`gdiGetDataValue`, see A.2.9.2, and

© ISO 2020 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO 20242-5:2020

`InformationReport` via callback in `refCBInformationReport`, see A.2.20.6) and there shall be no influence to the state of the workspace (`eNOT_USED`, `eUSED`). This application is called monitor application.

A monitor application shall have access to the existing trees of the RunTime side and to the data types of the communication parameters and operation parameters. An access to the Database side should be rejected, any access changing state or configuration shall be rejected with error `eOAD_OBJECT_ACCESS`.

For problems with data access, e.g. caused by access of the main application, the error `eOAD_DATAINUSE_OR_INCONSISTENT` shall be notified.

Exceptions shall be limited to errors occurring at the call of methods by the monitor application and method `gdiReleaseMonitorAccess` shall release the connection.

InformationReports for communication objects shall be sent to the monitor application only for those objects, which are processed for the main application. The object administration for sending events shall be specific for the main application. Therefore, monitor applications shall receive information reports only for objects,

— which have valid callbacks from the monitor application within the object tree and

— for which the InformationReport has been enabled by the main application.

An additional InformationReport is sent from the coordinator to the monitor, if the main application has processed a successful write or read of an attribute.

The global callback reference for InformationReports (parameter in `refCBInformationReport` of `gdiGetMonitorAccessByWorkspaceName`), shall be overwritten related to the attribute when method `GdiCoAttributeRT::gdiSetLocalCBInfReport` is used to define callbacks.

The number of connected monitor applications shall not be limited. Monitor applications shall stay valid if a workshop transfer is done with main applications.

If the main application wants to delete the workspace via `gdiDeleteWorkspace` while a monitor application is connected, the deletion is rejected with error `eINT_INVALID_ACCESS`.

5.5.6 Data types

All defined data types, including FO references, shall be supported. On the Description side, data type descriptions exist from the DCD. On RunTime side, data objects shall exist, which correspond to defined types. Complex types shall be created step-by-step in a tree structure.

The data exchange between application and device driver shall be mediated by the coordinator with providing formatted and allocated communication memory corresponding to the data type descriptions. Data exchange shall be done in two steps:

1. Data is copied form the source (application or driver) to the communication memory of the abstract data sink.

2. The data destination object reads the data from this communication memory.

APSI shall provide services for writing and reading of communication memory within the coordinator.

© ISO 2020 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO 20242-5:2020

Methods `gdiGet...Value` and `gdiSet...Value` (see e.g. B.2.4.1) shall handle the formatted access to the communication memory of the coordinator via streaming without launching any activity at the device driver. Further on, each sub-element of a complex data type should set the sub-element values separately (step by step).

After the communication memory has been updated, the data exchange with the device driver (via `GDI_Read` and `GDI_Write`) shall start using

`GdiCoExtendedObjectNavigator::gdiReadCommObjectData` and
`GdiCoExtendedObjectNavigator::gdiWriteCommObjectData`.

Alternatively, the interface `GdiCoCommObject` shall provide the execution of these two communication steps with only one method `gdiReadValue` or `gdiWriteValue` (i.e. update of the communication memory with application data and data exchange with the device driver). Figure 6 gives an overview on the data type handling of APSI.

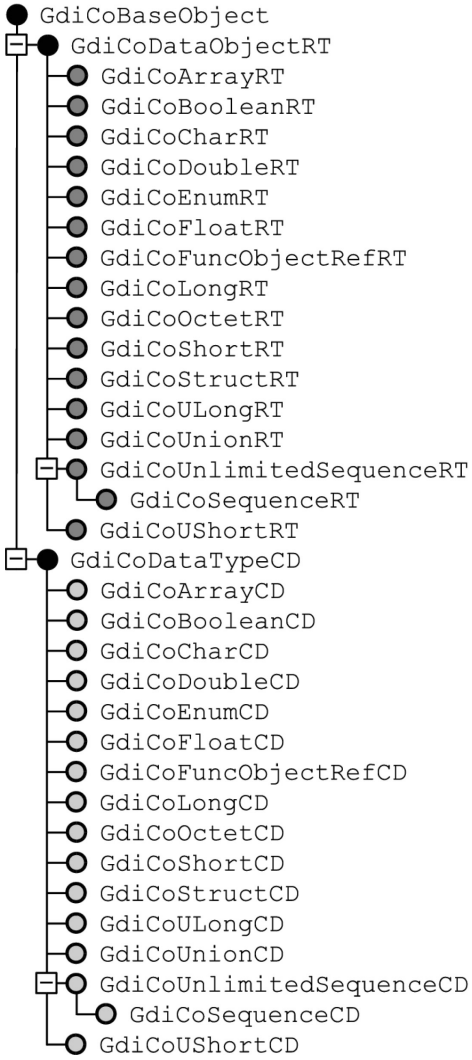


Figure 6 — Data type classes and instantiated data objects

5.5.7 Streaming

To enable the formatted application access also for complex data types, streaming shall be used for the data exchange between coordinator and application. Device driver specific data types defined in the DCD shall not be visible for the application.

© ISO 2020 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO 20242-5:2020

Stream objects shall be used for the data exchange between application and coordinator. The stream shall be a BLOB (Binary Large OBject), which is a sequence of bytes. A stream shall be set up according to the type descriptions in the DCD (see ISO 20242-4). The stream BLOB shall meet the alignment and byte order agreed upon for the respective data exchange (see coordinator datasheet in Annex I).

A non-typified streaming shall be used. No type description shall precede nor shall be transmitted separately or included.

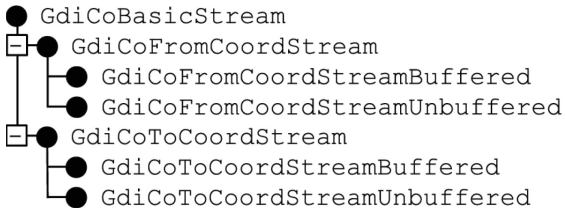


Figure 7 — Objects for streaming

Figure 7 shows objects defined for streaming in APSI. A stream object shall enable the stream based data exchange between application and coordinator. Possible implementations are, that the stream object either sets up the connection between two points (data source and sink) and thus provides for an unbuffered data exchange between these two points, or provides a container, which makes it possible to fill data into (package data) and hand over the package to any destination. The addressee may take the data from the package for further processing. This is a buffered data exchange. Buffered streaming is optional for this document.

Stream objects shall enable the platform independent data exchange between application and coordinator. The view of application to data therefore may be arbitrary.

With APSI, two kinds of stream objects are defined (see A.2.14 and A.2.26),

— `GdiCoFromCoordStream` and

— `GdiCoToCoordStream`,

which are derived from `GdiCoBasicStream`.

There are additional objects to distinguish between buffered and unbuffered data exchange (see A.2.15, A.2.16 and A.2.27, A.2.28).

An unbuffered streaming is only efficient, if alignment and byte order are identical between application and coordinator. If coordinator and application have different alignments and/or byte order, buffered data exchange should be used.

Using stream objects, the most different alignments and byte orders (see Annex G) may be used. The order of the serialized data within the stream object is determined by the kind of application. The coordinator has to be enabled to adjust to this order by passing on information on the order of the data in the stream object handed over. These context parameters are arguments for the construction of stream objects. The parameters shall precede with constants `cAppAlignment` and `cAppByteOrder`, defined in respective reference implementations (default constants as literal in the header). With the streaming the used alignment and byte order shall be decoded. Further rules for the application of stream objects are defined in Annex F.

© ISO 2020 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO 20242-5:2020

5.6 Error handling

Errors shall be reported with exceptions. It shall be distinguished between device driver errors (including device exceptions), which are passed to the application via coordinator, and coordinator errors. Device driver errors shall not be reflected in coordinator errors (see Figure 8).

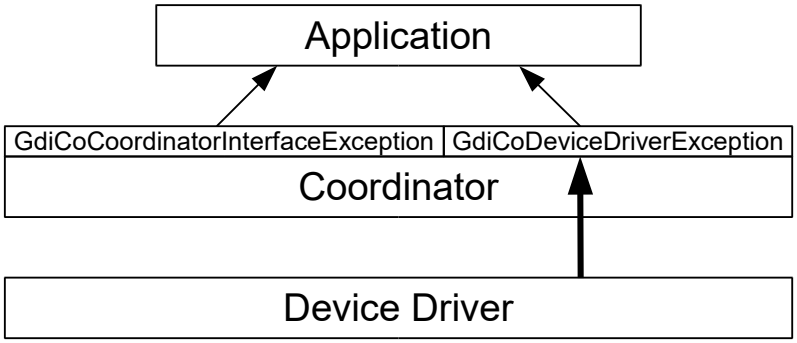


Figure 8 — Sources for coordinator and device driver errors

For details see A.2.7 to A.2.10.

Coordinator errors shall be assigned to one of the following classes:

— parameterization (faulty PID file),

— object administration (faulty use of APSI) and

— internal (coordinator, operating system, version, forbidden monitor access).

It is near at hand, that implementations of APSI contain additional internal data check features with respect to specific resources or internal limits. Accordingly support is not defined in APSI, but if a coordinator detects such an internal error, the error code `eINT_PRACTICAL_DATA_OUT_OF_RANGE` shall be used.

© ISO 2020 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO 20242-5:2020

Annex A
(normative)

Programmer's reference guide — Smart Access Interface

A.1 General

Single interfaces (respectively classes and their methods and attributes) of the Smart Access Interface, marked with «S», also shall belong to the Extended Access Interface, marked with «E» and the Full Access Interface, marked with «F». Therefore all single interfaces are marked with «S,E,F». The following detailed description is completed by Figures A.1 to A.31 for an overview on classes and their methods.

A.2 Detailed description of Small Access Interface

A.2.1.1 «S,E,F» GdiCoA_Version

This interface contains version information of the Application Program Service Interface.

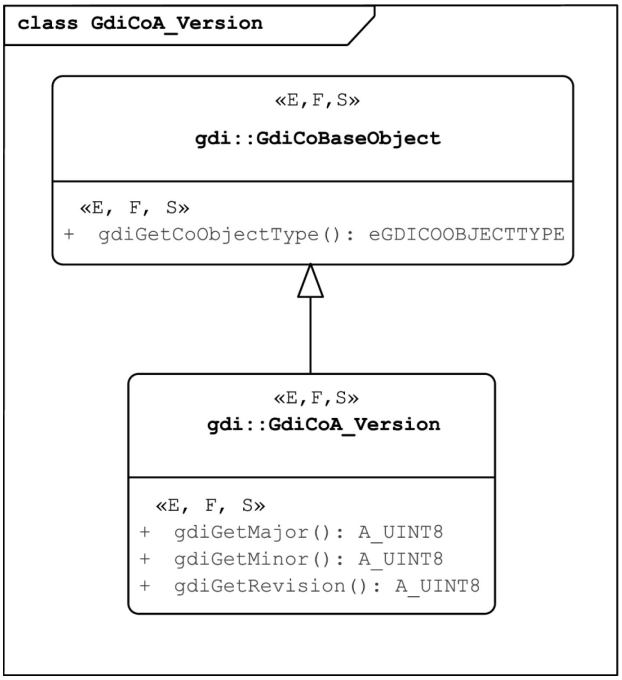


Figure A.1 — Hierarchical diagram of GdiCoA_Version

A.2.1.2 GdiCoA_Version :: «S,E,F» gdiGetMajor()

Function Call

```
gdiGetMajor
        (
        ) : A_UINT8
```

© ISO 2020 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO 20242-5:2020

Function Description

Returns the Major Version Number.

Return Value

Major Version Number.

A.2.1.3 GdiCoA_Version :: «S,E,F» gdiGetMinor()

Function Call

```
gdiGetMinor
         (
         ) : A_UINT8
```

Function Description

Returns the Minor Version number.

Return Value

Returns Minor Version number.

A.2.1.4 GdiCoA_Version :: «S,E,F» gdiGetRevision()

Function Call

```
gdiGetRevision
         (
         ) : A_UINT8
```

Function Description

Returns the Revision Number.

Return Value

Returns Revision Number.

A.2.2 «S,E,F» GdiCoAttributeRT

This interface contains services for the handling of attributes. These Communication Objects are distinguished by parameters, so that attributes may additionally exchange data by means of InformationReport and Accept, initiated by the driver.

© ISO 2020 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO 20242-5:2020

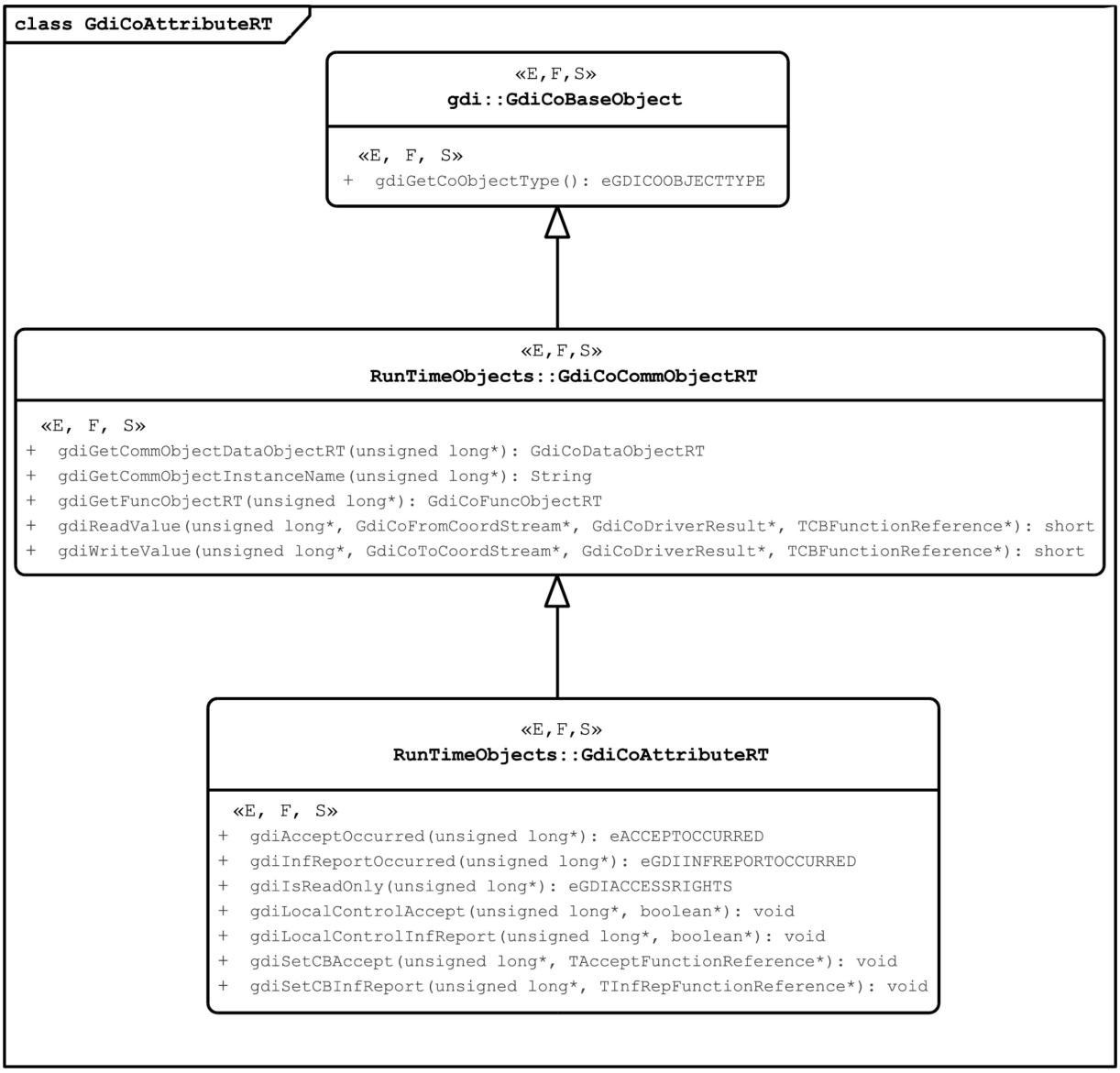


Figure A.2 — Hierarchical diagram of GdiCoAttributeRT

A.2.2.1 GdiCoAttributeRT :: «S,E,F» gdiAcceptOccurred()

Function Call

```
gdiAcceptOccurred
        (
        inout               unsigned long ulAppHnd
        /* in parameter
        Identifier of the application returned by gdiCreateWorkspace(...) or by
        gdiGetWorkspaceByName() */
        ) : eACCEPTOCCURRED
```

Function Description

Returns the information, if an Accept has been carried out since the last test or gdiWriteValue. The service is used, if no call back is supported by the application.

© ISO 2020 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO 20242-5:2020

Return Value

Returns eNOACCEPT (0) if no Accept occurred.
Returns eACCEPT (1) if Accept occurred.

A.2.2.2 GdiCoAttributeRT :: «S,E,F» gdiInfReportOccurred()

Function Call

```
gdiInfReportOccurred
        (
        inout               unsigned long ulAppHnd
        /* in parameter
        Identifier of the application returned by gdiCreateWorkspace(...) or by
        gdiGetWorkspaceByName() */
        ) : eGDIINFREPORTOCCURRED
```

Function Description

Returns the information, if an InformationReport has been carried out since the last test or gdiReadValue. The service is used, if no call back is supported by the application.

Return Value

Returns eNOINFREPORT (0) if no InormationReport occurred.
Returns eINFREPORT (1) if InformationReport occurred.

A.2.2.3 GdiCoAttributeRT :: «S,E,F» gdiIsReadOnly()

Function Call

```
gdiIsReadOnly
        (
        inout               unsigned long ulAppHnd
        /* in parameter
        Identifier of the application returned by gdiCreateWorkspace(...) or by
        gdiGetWorkspaceByName() */
        ) : eGDIACCESSRIGHTS
```

Function Description

Returns the information, if a Communication object is read only.

Return Value

Returns eREADWRITE (0) Read Write.
Returns eREADONLY (1) Read only.

A.2.2.4 GdiCoAttributeRT :: «S,E,F» gdiLocalControlAccept()

Function Call

```
gdiLocalControlAccept
        (
        inout               unsigned long ulAppHnd
        /* in parameter
        Identifier of the application returned by gdiCreateWorkspace(...) or by
        gdiGetWorkspaceByName() */
        inout               boolean bEnableControl
```


© ISO 2020 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO 20242-5:2020

```
        /* in parameter
        true = enable Accept Callbacks
        false = disable Accept Callbacks */
        ) : void
```

Function Description

Enable/disable the Accept Callback for this Communication Object.

Return Value

None

A.2.2.5 GdiCoAttributeRT :: «S,E,F» gdiLocalControlInfReport()

Function Call

```
gdiLocalControlInfReport
        (
        inout               unsigned long ulAppHnd
        /* in parameter
        Identifier of the application returned by gdiCreateWorkspace(...) or by
        gdiGetWorkspaceByName() */
        inout               boolean bEnableControl
        /* in parameter
        true = enable InformationReport
        false = disable InformationReport */
        ) : void
```

Function Description

Enable or disable the InformationReport CallBack for this attribute.

If the InformationReport flag is enabled at the GdiCoAttributRT and the device driver does not support information reports for this attribute, the coordinator shall send an InformationReport to the monitoring client any time the attribute is changed by another client.

If the device driver sends an InformationReport to this attribute, the information shall be delivered to all monitoring clients which have enabled the InformationReport flag.

Return Value

None

A.2.2.6 GdiCoAttributeRT :: «S,E,F» gdiSetCBAccept()

Function Call

```
gdiSetCBAccept
        (
        inout               unsigned long ulAppHnd
        /* in parameter
        Identifier of the application returned by gdiCreateWorkspace(...) or by
        gdiGetWorkspaceByName() */
        inout               TAcceptFunctionReference refCBAccept
        /* in parameter
        Reference to a call back method which handles the accept mechanism. A
        NIL reference signals that no call back is supported or the global
        callback reference is to use.
```

© ISO 2020 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO 20242-5:2020

```
        At the coordinator call back to the application, an reference to the
        specific GdiCoCommObjectRT is passed. */
        ) : void
```

Function Description

Register a local Accept callback for this communication object. The global call back (given by gdiCreateWorkspace) shall be deactivated.

Return Value

None

A.2.2.7 GdiCoAttributeRT :: «S,E,F» gdiSetCBInfReport()

Function Call

```
gdiSetCBInfReport
        (
        inout               unsigned long ulAppHnd
        /* in parameter
        Identifier of the application returned by gdiCreateWorkspace(...) or by
        gdiGetWorkspaceByName() */
        inout               TInfRepFunctionReference refCBInformationReport
        /* in parameter
        Reference to a call back method which handles the informationreport
        mechanism. A NIL reference signals that no call back is supported or the
        global callback reference is to use.
        At the coordinator call back to the application, an reference to the
        specific GdiCoCommObjectRT is passed. */
        ) : void
```

Function Description

Register a local InformationReport callback for this communication object. The global call back (given by gdiCreateWorkspace) shall be deactivated.

If the InformationReport flag is enabled at the GdiCoAttributRT and the device driver does not support information reports for this attribute, the coordinator shall send an InformationReport to the monitoring client any time the attribute is changed by another client.

If the device driver sends an InformationReport to this attribute, the information will be delivered to all monitoring clients, which have enabled the InformationReport flag.

Return Value

None

A.2.3 «S,E,F» GdiCoBaseObject

This interface is the base interface of all coordinator objects. This services occurs in inheritance only.

© ISO 2020 – All rights reserved
Documents Document Sharing Hub

Full document details: BS ISO 20242-5:2020



Figure A.3 — Hierarchical diagram of GdiCoBaseObject

A.2.3.1 GdiCoBaseObject :: «S,E,F» gdiGetCoObjectType()

Function Call

```
gdiGetCoObjectType
        (
        ) : eGDICOOBJECTTYPE
```

Function Description

Returns the enumeration for detecting the type of the Coordinator Object reference.

Return Value

Returns a value of GDICOOBJECTTYPE.

A.2.4 «S,E,F» GdiCoBasicStream

This interface is the base interface of the stream objects. These services only occur in inheritance.

For monitor access the data is duplicated to all clients.

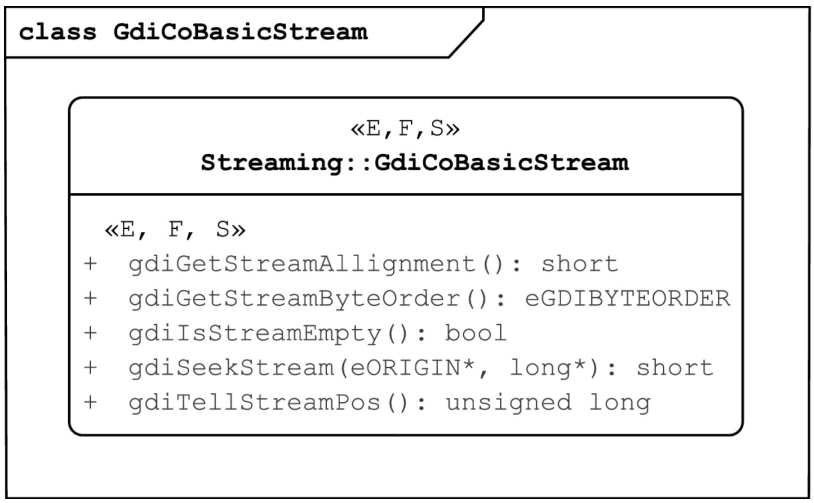


Figure A.4 — Hierarchical diagram of GdiCoBasicStream

© ISO 2020 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO 20242-5:2020

A.2.4.1 GdiCoBasicStream :: «S,E,F» gdiGetStreamAllignment()

Function Call

```
gdiGetStreamAllignment
        (
        ) : short
```

Function Description

Returns the Allignment of the Data in the Stream Object.

Return Value

Returns the Allignment of the Data in the Stream Object.

A.2.4.2 GdiCoBasicStream :: «S,E,F» gdiGetStreamByteOrder()

Function Call

```
gdiGetStreamByteOrder
        (
        ) : eGDIBYTEORDER
```

Function Description

Returns the ByteOrder of the Data in the Stream.

Return Value

Returns the ByteOrder of the Data in the Stream.

A.2.4.3 GdiCoBasicStream :: «S,E,F» gdiIsStreamEmpty()

Function Call

```
gdiIsStreamEmpty
        (
        ) : bool
```

Function Description

This method is used to check the type compatibly of the Date in stream against the Data in the Application.

If the return value is true after a communication, the type is with a high probability correctly interpreted.

(similar to "End Of File" Flag)

Return Value

true = Stream is empty.
false = Stream is not empty.

© ISO 2020 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO 20242-5:2020

A.2.4.4 GdiCoBasicStream :: «S,E,F» gdiSeekStream()

Function Call

```
gdiSeekStream
        (
        inout               eORIGIN eOriginator
        /* in parameter
        contains the initial position. */
        inout               long lPos
        /* in parameter
        contains the position in which byte to move the stream pointer from
        origin. */
        ) : short
```

Function Description

Moves the stream pointer in a stream to a specific location.

Return Value

0 : sucessful seeked
!= 0 : error

A.2.4.5 GdiCoBasicStream :: «S,E,F» gdiTellStreamPos()

Function Call

```
gdiTellStreamPos
        (
        ) : unsigned long
```

Function Description

Report current position in output stream.

Return Value

Returns the current position in the stream.

A.2.5 «S,E,F» GdiCoCommObjectIterator

This interface contains the service for the application of a CommObject iterator, sequential output of all Communication Objects which have been set up within a Function Object.

© ISO 2020 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO 20242-5:2020

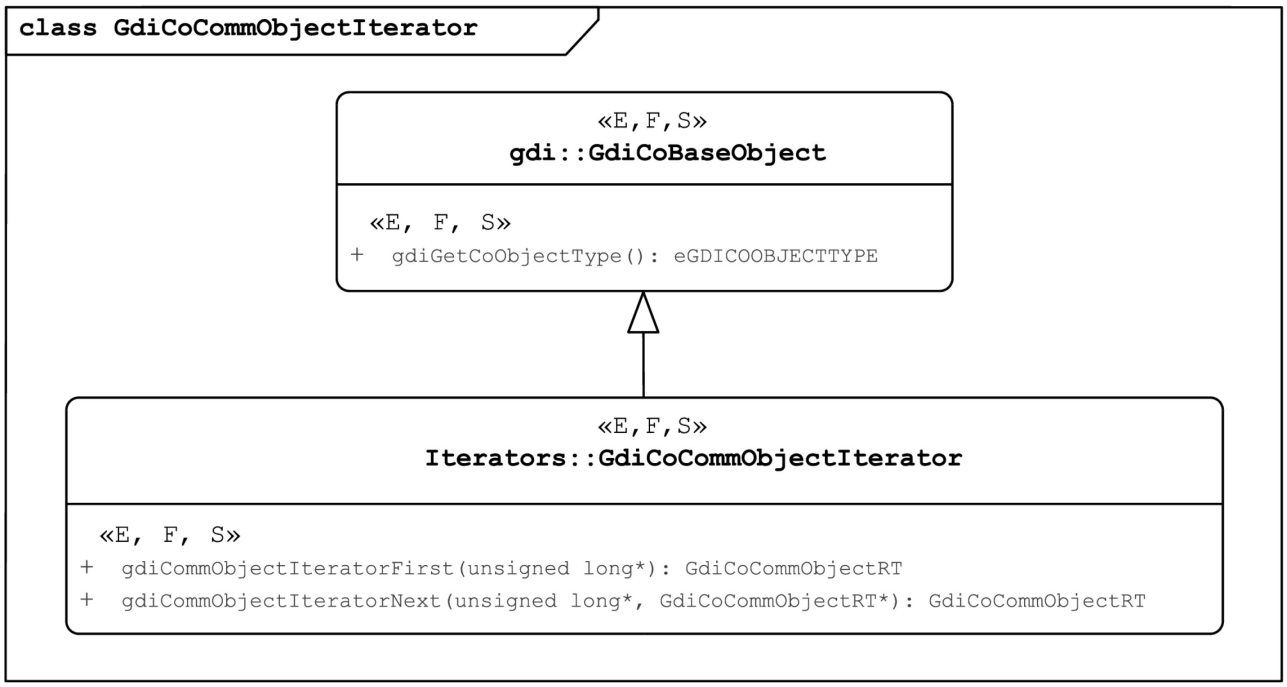


Figure A.5 — Hierarchical diagram of GdiCoCommObjectIterator

A.2.5.1 GdiCoCommObjectIterator :: «S,E,F» gdiCommObjectIteratorFirst()

Function Call

```
gdiCommObjectIteratorFirst
        (
        inout               unsigned long ulAppHnd
        /* in parameter
        Identifier of the application returned by gdiCreateWorkspace(...) or by
        gdiGetWorkspaceByName() */
        ) : GdiCoCommObjectRT
```

Function Description

Sets the internal pointer to the first ComObject within the Workspace.

Return Value

Returns a reference to the first GdiCoCommObject if it exists, otherwise a NIL Reference returns.

A.2.5.2 GdiCoCommObjectIterator :: «S,E,F» gdiCommObjectIteratorNext()

Function Call

```
gdiCommObjectIteratorNext
        (
        inout               unsigned long ulAppHnd
        /* in parameter
        Identifier of the application returned by gdiCreateWorkspace(...) or by
        gdiGetWorkspaceByName() */
        inout               GdiCoCommObjectRT refLastElement
        /* in parameter
```


© ISO 2020 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO 20242-5:2020

```
A reference to a element returned by gdiComObjectIteratorFirst or
gdiComObjectIteratorNext. */
) : GdiCoCommObjectRT
```

Function Description

Detects the reference to the current ComObject (Parameter) within the function object. Sets the internal pointer to the next ComObject within the FO (if existing) and returns the reference.

Return Value

Returns reference to a coordinator CommObject (Administration unit within the coordinator).

A.2.6 «S,E,F» GdiCoCommObjectRT

This interface contains services for the use of Communication Objects. These services only occur in inheritance.

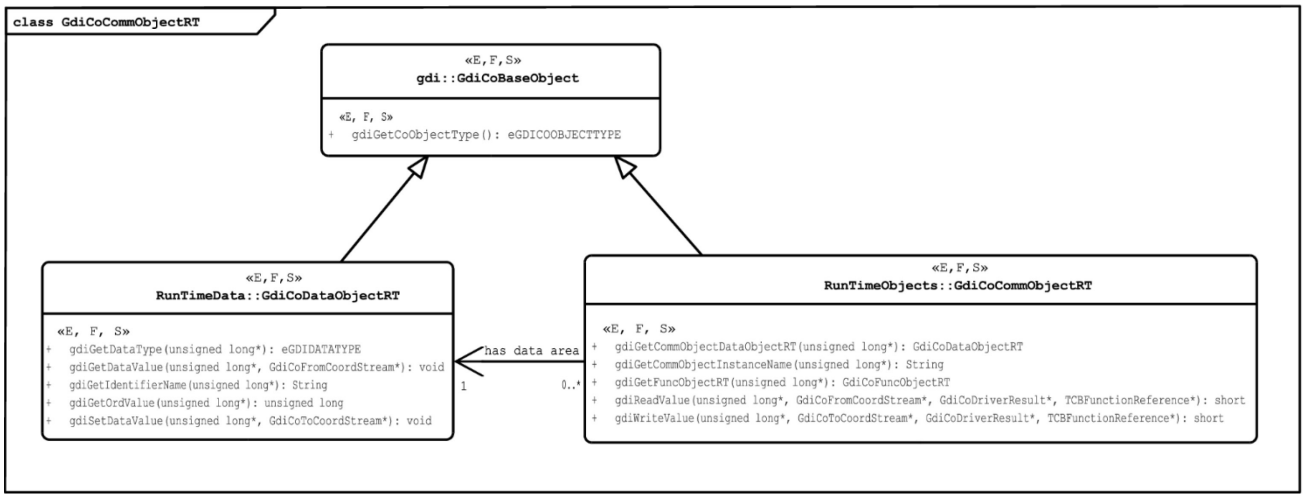


Figure A.6 — Hierarchical diagram of GdiCoCommObjectRT

A.2.6.1 GdiCoCommObjectRT :: «S,E,F» gdiGetCommObjectDataObjectRT()

Function Call

```
gdiGetCommObjectDataObjectRT
        (
        inout               unsigned long ulAppHnd
        /* in parameter
        Identifier of the application returned by gdiCreateWorkspace(...) or by
        gdiGetWorkspaceByName() */
        ) : GdiCoDataObjectRT
```

Function Description

Returns the reference to the corresponding data object of the given communication object.

Return Value

Returns the reference to the corresponding data object.

© ISO 2020 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO 20242-5:2020

A.2.6.2 GdiCoCommObjectRT :: «S,E,F» gdiGetCommObjectInstanceName()

Function Call

```
gdiGetCommObjectInstanceName
        (
        inout               unsigned long ulAppHnd
        /* in parameter
        Identifier of the application returned by gdiCreateWorkspace(...) or by
        gdiGetWorkspaceByName() */
        ) : String
```

Function Description

Returns the instance name of the object defined during parameterization.

Only one ComObject instance is possible so the attribute/parameter name inside the interface and object name can be identical.

Return Value

Returns the name of the object.

A.2.6.3 GdiCoCommObjectRT :: «S,E,F» gdiGetFuncObjectRT()

Function Call

```
gdiGetFuncObjectRT
        (
        inout               unsigned long ulAppHnd
        /* in parameter
        Identifier of the application returned by gdiCreateWorkspace(...) or by
        gdiGetWorkspaceByName() */
        ) : GdiCoFuncObjectRT
```

Function Description

Creates reference to the existing Function Object.

The referenced Function Object contains this Communication Object. (used to evaluate the corresponding Function Object to the Communication Object delivered by an Information Report).

Return Value

Returns the reference to the GdiCoFuncObjectRT if succesful.

A.2.6.4 GdiCoCommObjectRT :: «S,E,F» gdiReadValue()

Function Call

```
gdiReadValue
        (
        inout               unsigned long ulAppHnd
        /* in parameter
        Identifier of the application returned by gdiCreateWorkspace(...) or by
        gdiGetWorkspaceByName() */
        inout               GdiCoFromCoordStream refDestValue
        /* out parameter
```


© ISO 2020 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO 20242-5:2020

```
Reference to a stream object, the type is identical to the type of the
Communication object. */
inout               GdiCoDriverResult refDestInfo
/* out parameter
contains the reference to the Information Object, warnings and
informations of the Driver will be stored in it. */
inout               TCBFunctionReference refCBCompleteRead
/* in parameter
Reference to a call back method which handles the read mechanism. A NIL
reference signals a synchronous call. */
) : short
```

Function Description

Reads a Communication object from the driver and returns a data stream by reference. The data in the stream is a sequence of bytes.

Return Value

Describes the executed communication by the GDI driver:
0 = synchronous return;
1 = asynchronous return.

A.2.6.5 GdiCoCommObjectRT :: «S,E,F» gdiWriteValue()

Function Call

```
gdiWriteValue
        (
        inout               unsigned long ulAppHnd
        /* in parameter
        Identifier of the application returned by gdiCreateWorkspace(...) or by
        gdiGetWorkspaceByName() */
        inout               GdiCoToCoordStream refSourceValue
        /* in parameter
        Reference to a stream object, the type is identical to the type of the
        Communication object. */
        inout               GdiCoDriverResult refDestInfo
        /* out parameter
        contains the reference to the Information Object, warnings and
        informations of the Driver will be stored in it. */
        inout               TCBFunctionReference refCBCompleteWrite
        /* in parameter
        Reference to a call back method which handles the write mechanism. A NIL
        reference signals synchronous call. */
        ) : short
```

Function Description

Writes a data value (by reference) of a communication object to the coordinator and starts the transfer to the driver.

Return Value

Describes the executed communication by the GDI driver:
0 = synchronous return;
1 = asynchronous return.

© ISO 2020 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO 20242-5:2020

A.2.7 «S,E,F» GdiCoCoordinatorException

This is the base error class and shall be used for error handling. See also 5.6 for the separated processing of coordinator internal errors and device driver errors.

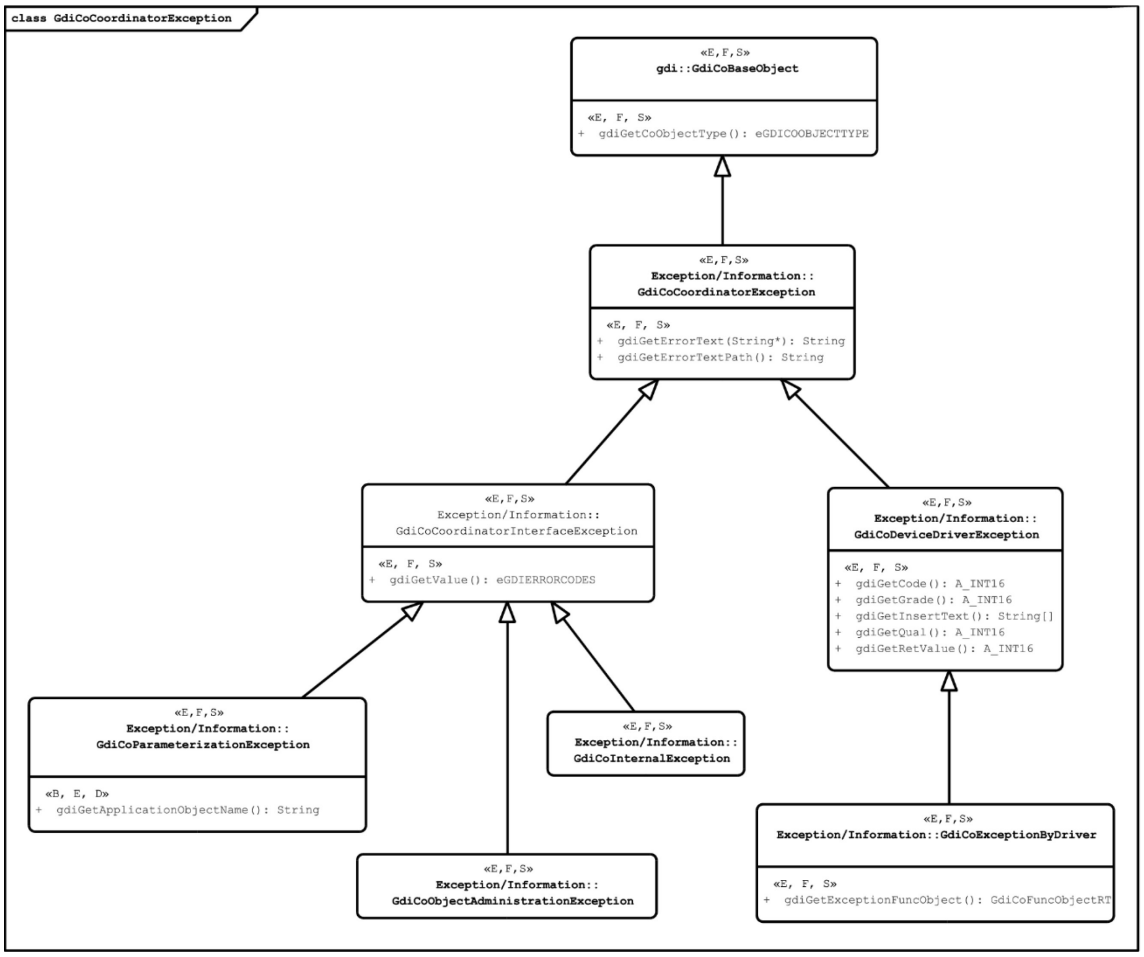


Figure A.7 — Hierarchical diagram of GdiCoCoordinatorException

A.2.7.1 GdiCoCoordinatorException :: «S,E,F» gdiGetErrorText()

Function Call

```
gdiGetErrorText
        (
        inout                String strLanguage
        /* in parameter
        requested language token according to ISO 639. */
        ) : String
```

Function Description

Returns the corresponding String in the corresponding Language including the replace text from the standard DIT if exist.

Otherwise an empty String is returned.

© ISO 2020 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO 20242-5:2020

Return Value

Returns the corresponding String in the corresponding Language including the replace text from the standard DIT if exist.

Otherwise an empty String is returned.

A.2.7.2 GdiCoCoordinatorException :: «S,E,F» gdiGetErrorTextPath()

Function Call

```
gdiGetErrorTextPath
        (
        ) : String
```

Function Description

Returns a path reference as String to a DIT or DII to the corresponding error message.

This path can be used to ask an external DIT/DII error server for the language specific error message.

Return Value

Returns a path reference as String to a DIT or DII to the corresponding error message.

This path can be used to ask an external DIT/DII error server for the language specific error message.

A.2.8 «S,E,F» GdiCoCoordinatorInterfaceException

This exception class shall be thrown if coordinator specific errors occur. Details of errors shall be described with child classes.

© ISO 2020 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO 20242-5:2020

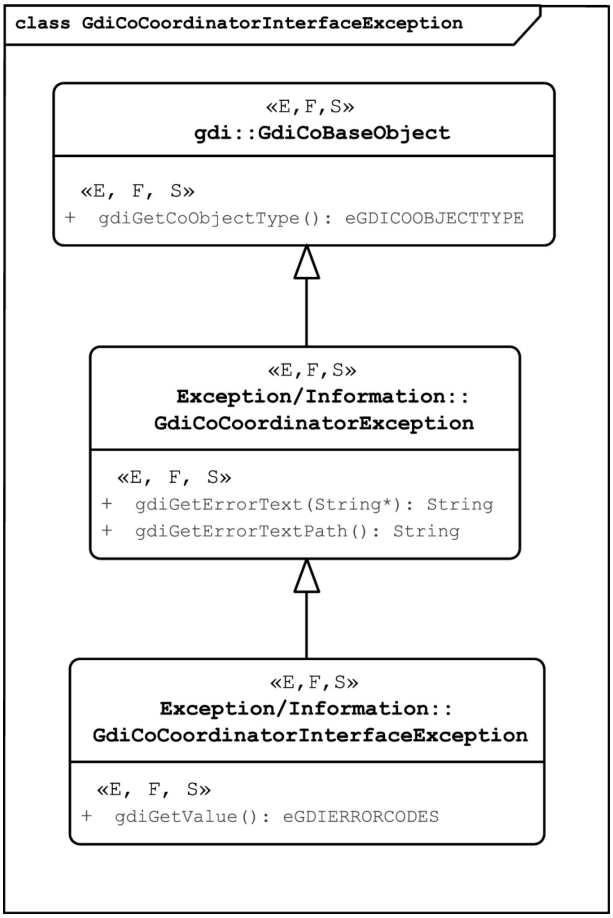


Figure A.8 — Hierarchical diagram of GdiCoCoordinatorInterfaceException

A.2.8.1 GdiCoCoordinatorInterfaceException :: «S,E,F» gdiGetValue()

Function Call

```
gdiGetValue
        (
        ) : eGDIERRORCODES
```

Function Description

Returns the error Value as a number.

See the eGDIERRORCODES enumeration for details.

Return Value

Returns the error codes as defined in the eGDIERRORCODES enumeration.

A.2.8.2 GdiCoDataObjectRT :: «S,E,F» gdiGetOrdValue()

Function Call

```
gdiGetOrdValue
        (
        inout               unsigned long ulAppHnd
```


© ISO 2020 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO 20242-5:2020

```
        /* in parameter
        Identifier of the application returned by gdiCreateWorkspace(...) or by
        gdiGetWorkspaceByName() */
        ) : unsigned long
```

Function Description

Returns the internal position value of the type (sub type) in a structure or union. Is used for the detection of the switch value of a sub type within a union. Within a structure this value is used for information only. If the type is a sub type in sequences or arrays, this service returns the value 0.

Return Value

Returns the internal position of this type as sub type within a structure or union.

A.2.8.3 GdiCoDataObjectRT :: «S,E,F» gdiSetDataValue()

Function Call

```
gdiSetDataValue
        (
        inout               unsigned long ulAppHnd
        /* in parameter
        Identifier of the application returned by gdiCreateWorkspace(...) or by
        gdiGetWorkspaceByName() */
        inout               GdiCoToCoordStream refSourceDataStream
        /* in parameter
        Contains the reference to the source Data Stream. */
        ) : void
```

Function Description

Sets the value of a data area within the Coordinator with the value handed over by the stream reference. The data are handled as a block of Bytes. The number of Bytes is determined by the type of the datum.

Return Value

None

A.2.9 «S,E,F» GdiCoDataObjectRT

This interface contains services for the access of data within the coordinator.

© ISO 2020 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO 20242-5:2020



Figure A.9 — Hierarchical diagram of GdiCoDataObjectRT

A.2.9.1 GdiCoDataObjectRT :: «S,E,F» gdiGetDataType()

Function Call

```
gdiGetDataType
        (
        inout               unsigned long ulAppHnd
        /* in parameter
        Identifier of the application returned by gdiCreateWorkspace(...) or by
        gdiGetWorkspaceByName() */
        ) : eGDIDATATYPE
```

Function Description

Returns the type as a numeric value (enum).

Return Value

Returns types as a numeric value (described in Semantics).

A.2.9.2 GdiCoDataObjectRT :: «S,E,F» gdiGetDataValue()

Function Call

```
gdiGetDataValue
```


© ISO 2020 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO 20242-5:2020

```
(
inout               unsigned long ulAppHnd
/* in parameter
Identifier of the application returned by gdiCreateWorkspace(...) or by
gdiGetWorkspaceByName() */
inout               GdiCoFromCoordStream refDestDataStream
/* in parameter
Contains the reference to the destination Data Stream in the application.
*/
) : void
```

Function Description

Returns the value of a data area within the Coordinator with as stream reference. The data are handled as a block of Bytes. The number of Bytes is determined by the type of the datum.

Return Value

None

A.2.9.3 GdiCoDataObjectRT :: «S,E,F» gdiGetIdentifierName()

Function Call

```
gdiGetIdentifierName
        (
        inout               unsigned long ulAppHnd
        /* in parameter
        Identifier of the application returned by gdiCreateWorkspace(...) or by
        gdiGetWorkspaceByName() */
        ) : String
```

Function Description

Returns the name of the type. Is used for the identification of derived types or sub-types.

Return Value

Returns a text string (Pointer, class, address of sz area).

A.2.10 «S,E,F» GdiCoDeviceDriverException

This shall be used for throwing exceptions caused by device driver errors.

© ISO 2020 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO 20242-5:2020

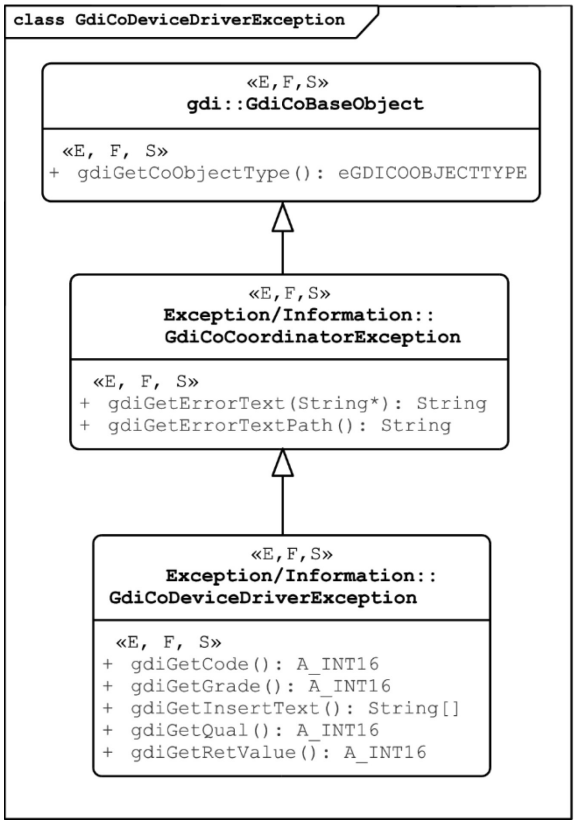


Figure A.10 — Hierarchical diagram of GdiCoDeviceDriverException

A.2.10.1 GdiCoDeviceDriverException :: «S,E,F» gdiGetCode()

Function Call

```
gdiGetCode
          (
          ) : A_INT16
```

Function Description

Returns the corresponding value of the GDI API error.

Return Value

Returns the errorcode of a gdi device driver api function.

A.2.10.2 GdiCoDeviceDriverException :: «S,E,F» gdiGetGrade()

Function Call

```
gdiGetGrade
          (
          ) : A_INT16
```

Function Description

Returns the corresponding value of the GDI API error.

© ISO 2020 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO 20242-5:2020

Return Value

Returns the errorcode of a GDI device driver API function.

A.2.10.3 GdiCoDeviceDriverException :: «S,E,F» gdiGetInsertText()

Function Call

```
gdiGetInsertText
        (
        ) : String[]
```

Function Description

Returns Replace text from GDI Driver as list of strings.

Return Value

Returns a list of String representing the replace text from GDI Driver.

A.2.10.4 GdiCoDeviceDriverException :: «S,E,F» gdiGetQual()

Function Call

```
gdiGetQual
        (
        ) : A_INT16
```

Function Description

Returns the corresponding value of the Devic Driver API error.

Return Value

Returns the errorcode of a GDI device driver API function.

A.2.10.5 GdiCoDeviceDriverException :: «S,E,F» gdiGetRetValue()

Function Call

```
gdiGetRetValue
        (
        ) : A_INT16
```

Function Description

Returns the corresponding value of the GDI API error.

Return Value

Returns the errorcode of a GDI device driver API function.

A.2.11 «S,E,F» GdiCoDriverResult

This interface shall transport warnings and informations from the device driver.

© ISO 2020 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO 20242-5:2020

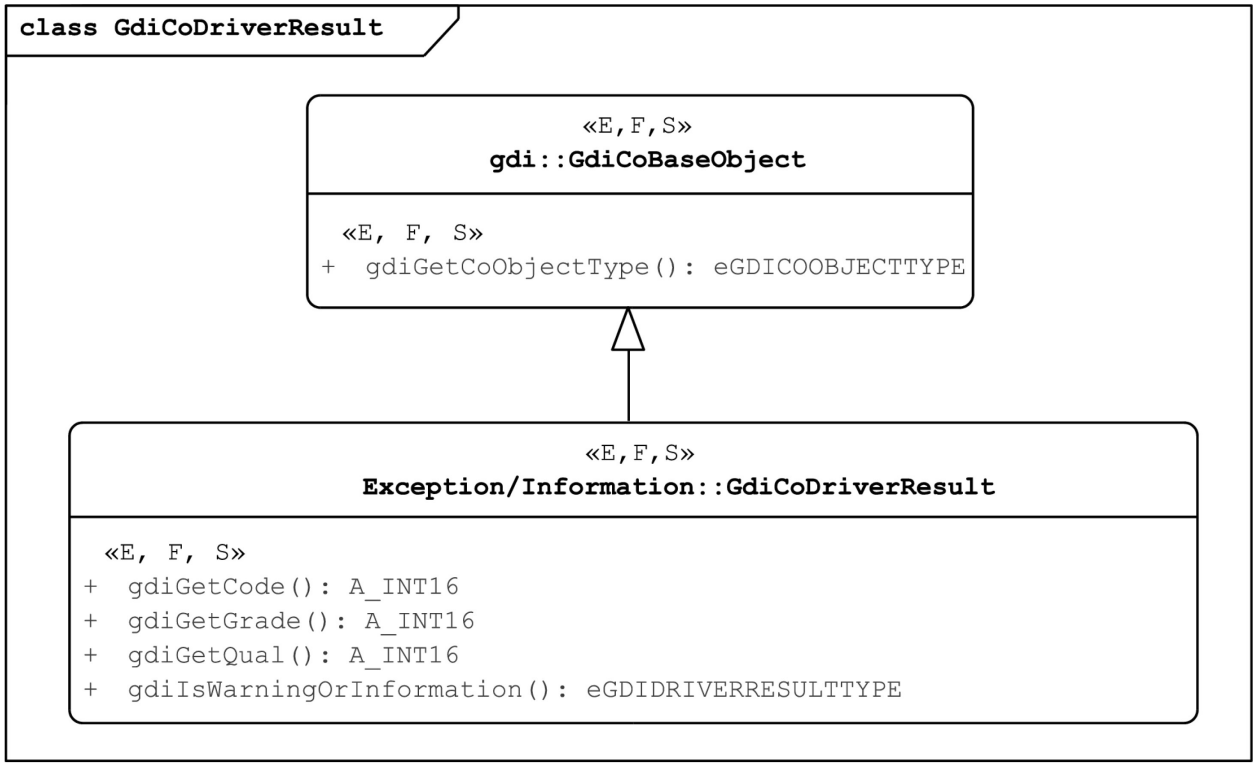


Figure A.11 — Hierarchical diagram of GdiCoDriverResult

A.2.11.1 GdiCoDriverResult :: «S,E,F» gdiGetCode()

Function Call

```
gdiGetCode
        (
        ) : A_INT16
```

Function Description

Returns the corresponding value of the GDI API information.

Return Value

Returns the information of a GDI device driver API function.

A.2.11.2 GdiCoDriverResult :: «S,E,F» gdiGetGrade()

Function Call

```
gdiGetGrade
        (
        ) : A_INT16
```

Function Description

Returns the corresponding value of the GDI API information.

© ISO 2020 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO 20242-5:2020

Return Value

Returns the information code of a GDI device driver API function.

A.2.11.3 GdiCoDriverResult :: «S,E,F» gdiGetQual()

Function Call

```
gdiGetQual
        (
        ) : A_INT16
```

Function Description

Returns the corresponding value of the GDI API information.

Return Value

Returns the information code of a GDI device driver API function.

A.2.11.4 GdiCoDriverResult :: «S,E,F» gdiIsWarningOrInformation()

Function Call

```
gdiIsWarningOrInformation
        (
        ) : eGDIDRIVERRESULTTYPE
```

Function Description

Returns the enumeration about the information, if a warning or a information has occurred.

Return Value

Returns the enumeration about the information, if a warning or a information has occurred.

A.2.12 «S,E,F» GdiCoElementIterator

This interface shall contain the service for the application of a type iterator, sequential output of all sub types which have been set up within a structure or union.

© ISO 2020 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO 20242-5:2020

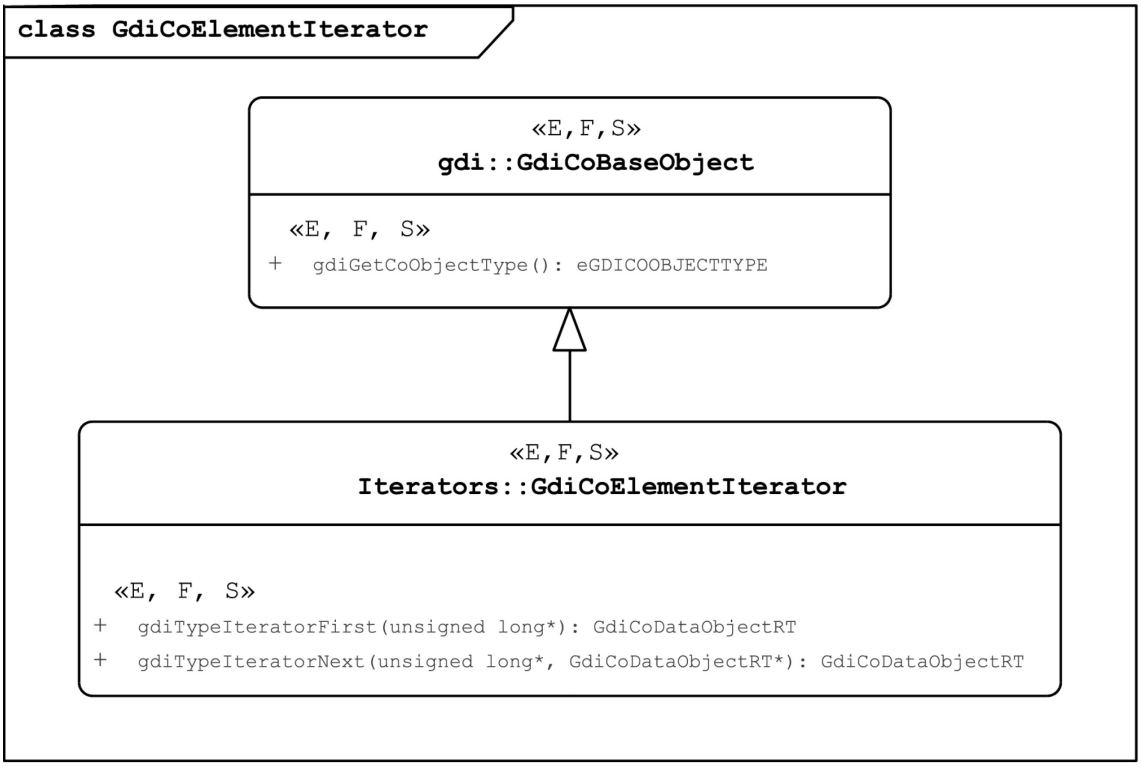


Figure A.12 — Hierarchical diagram of GdiCoElementIterator

A.2.12.1 GdiCoElementIterator :: «S,E,F» gdiTypeIteratorFirst()

Function Call

```
gdiTypeIteratorFirst
        (
        inout               unsigned long ulAppHnd
        /* in parameter
        Identifier of the application returned by gdiCreateWorkspace(...) or by
        gdiGetWorkspaceByName() */
        ) : GdiCoDataObjectRT
```

Function Description

Sets the internal pointer to the first Type within the Control interface.

Return Value

Returns a reference to the first GdiCoBaseType if it exists, otherwise a NIL Reference returns.

A.2.12.2 GdiCoElementIterator :: «S,E,F» gdiTypeIteratorNext()

Function Call

```
gdiTypeIteratorNext
        (
        inout               unsigned long ulAppHnd
        /* in parameter
```


© ISO 2020 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO 20242-5:2020