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)

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

Contents

Page

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, *Interoperability, 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.

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.

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/

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]

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).

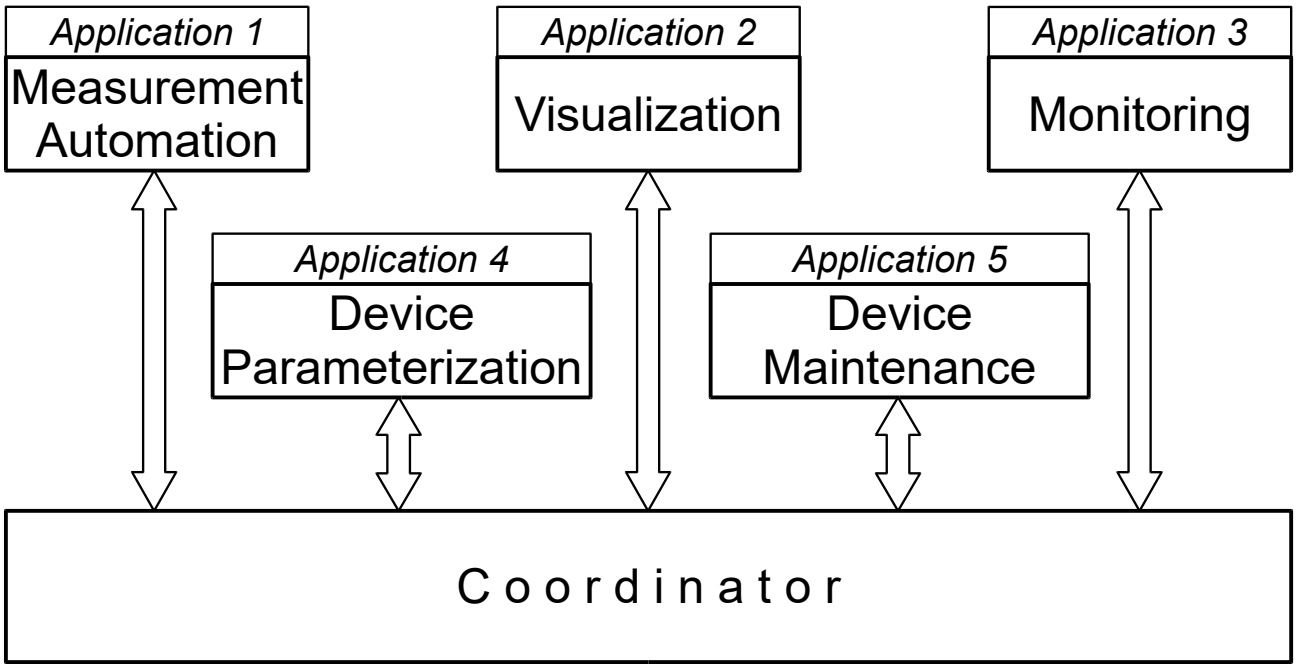

Figure 1 — Exemplary application programs using a coordinator

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.

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.

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.

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).

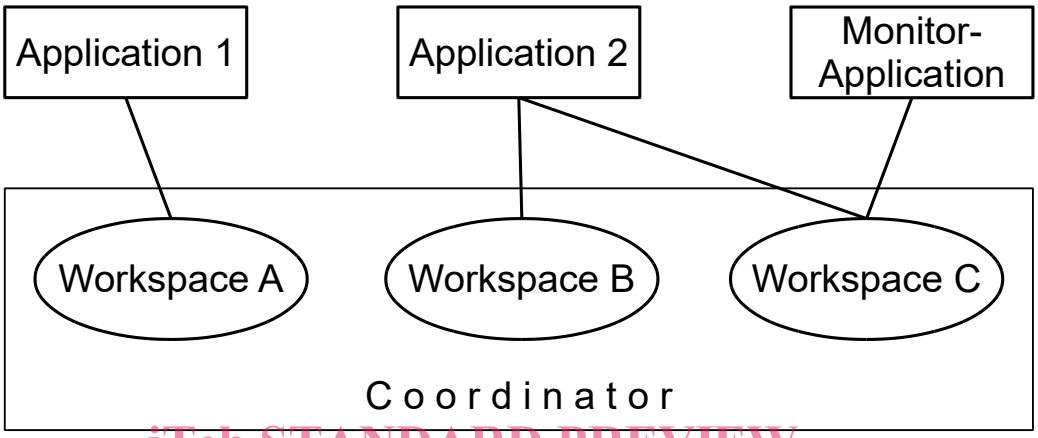

Figure 2 — Applications and workspaces

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.

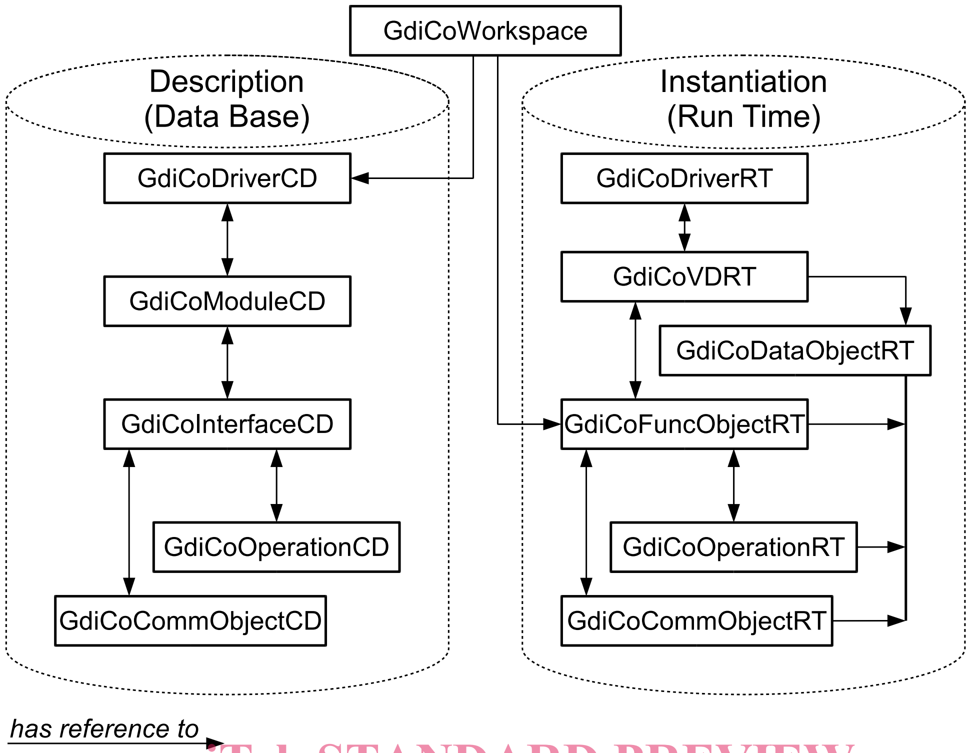

Figure 3 — Workspace view for service organisation

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.

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.