IEC 62769-6

Edition 2.0 2021-02
REDLINE VERSION

INTERNATIONAL STANDARD

colour inside

Field device integration (FDI) –
Part 6: ~~FDI~~ Technology Mapping

IEC 62769-6:2021-02 RLV(en)

IEC 62769-6

Edition 2.0 2021-02
REDLINE VERSION

INTERNATIONAL STANDARD

colour inside

Field device integration (FDI) – Part 6: ~~FDI~~ Technology Mapping

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

ICS 25.040.40; 35.100.05

ISBN 978-2-8322-9398-0

Warning! Make sure that you obtained this publication from an authorized distributor.

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

____

FIELD DEVICE INTEGRATION (FDI) –

Part 6: ~~FDI~~ Technology Mapping

FOREWORD

This redline version of the official IEC Standard allows the user to identify the changes made to the previous edition IEC 62769-6:2015. A vertical bar appears in the margin wherever a change has been made. Additions are in green text, deletions are in strikethrough red text.

International Standard IEC 62769-6 has been prepared by subcommittee 65E: Devices and integration in enterprise systems, of IEC technical committee 65: Industrial-process measurement, control and automation.

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

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

a) redesign of the security concept for UIP execution.

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

FDIS	Report on voting
65E/763/FDIS	65E/773/RVD

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

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

A list of all parts in the IEC 62769 series, published under the general title *Field Device Integration (FDI)*, can be found on the IEC website.

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

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- withdrawn,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

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

INTRODUCTION

~~The International Electrotechnical Commission (IEC) draws attention to the fact that it is claimed that compliance with this document may involve the use of patents concerning~~

~~a) Method for the Supplying and Installation of Device-Specific Functionalities, see Patent Family DE10357276;~~

~~b) Method and device for accessing a functional module of automation system, see Patent Family EP2182418;~~

~~c) Methods and apparatus to reduce memory requirements for process control system software applications, see Patent Family US2013232186;~~

~~d) Extensible Device Object Model, see Patent Family US12/893,680.~~

~~IEC takes no position concerning the evidence, validity and scope of this patent right.~~

~~The holders of these patent rights have assured the IEC that he/she is willing to negotiate licences either free of charge or under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of this patent right is registered with IEC. Information may be obtained from:~~

~~a) ABB Research Ltd~~
~~Claes Rytoft~~
~~Affolterstrasse 4~~
~~Zurich, 8050~~
~~Switzerland~~

~~b) Phoenix Contact GmbH & Co KG~~
~~Intellectual Property, Licenses & Standards~~
~~Flachsmarktstrasse 8, 32825 Blomberg~~
~~Germany~~

~~c) Fisher Controls International LLC~~
~~John Dilger, Emerson Process Management LLLP~~
~~301 S. 1st Avenue, Marshaltown, Iowa 50158~~
~~USA~~

~~d) Rockwell Automation Technologies, Inc.~~
~~1 Allen-Bradley Drive~~
~~Mayfield Heights, Ohio 44124~~
~~USA~~

~~Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those identified above. IEC shall not be held responsible for identifying any or all such patent rights.~~

~~ISO (www.iso.org/patents) and IEC (http://patents.iec.ch) maintain on-line data bases of patents relevant to their standards. Users are encouraged to consult the data bases for the most up to date information concerning patents.~~

The IEC 62769 series has the general title *Field Device Integration (FDI)* and the following parts:

– Part 1: Overview
– Part 2: FDI Client
– Part 3: FDI Server
– Part 4: FDI Packages
– Part 5: FDI Information Model
– Part 6: FDI Technology Mapping
– Part 7: FDI Communication Devices

- Part 100: Profiles – Generic Protocol Extensions
- Part 101-1: Profiles – Foundation Fieldbus H1
- Part 101-2: Profiles – Foundation Fieldbus HSE
- Part 103-1: Profiles – PROFIBUS
- Part 103-4: Profiles – PROFINET
- Part 109-1: Profiles – HART and WirelessHART
- Part 115-2: Profiles – Protocol-specific Definitions for Modbus RTU
- Part 150-1: Profiles – ISA 100.11a

FIELD DEVICE INTEGRATION (FDI) –

Part 6: ~~FDI~~ Technology Mapping

1 Scope

This part of IEC 62769 specifies the technology mapping for the concepts described in the Field Device Integration (FDI) standard. The technology mapping focuses on implementation regarding the components FDI Client and User Interface Plug-in (UIP) that are specific only to the WORKSTATION platform/.NET as defined in IEC 62769-4~~:2015, Annex E~~.

2 Normative references

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

IEC 61804 (all parts), *Function blocks (FB) for process control and Electronic Device Description Language (EDDL)*

IEC 62769-1, *Field Device Integration (FDI) – Part 1: Overview*

IEC 62769-2, *Field Device Integration (FDI) – Part 2: FDI Client*

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

IEC 62541 (all parts), *OPC Unified Architecture*

~~IEC 61804 (all parts), *Function blocks (FB) for process control*~~

~~IEC 62769-1, *Field Device Integration (FDI) – Part 1: Overview*~~

~~NOTE IEC 62769-1 is technically identical to FDI-2021.~~

~~IEC 62769-2, *Field Device Integration (FDI) – Part 2: FDI Client*~~

~~NOTE 1 IEC 62769-2 is technically identical to FDI-2022.~~

~~NOTE 2 IEC 62769-2 is technically identical to FDI-2023.~~

~~IEC 62769-4:2015, *Field Device Integration (FDI) – Part 4: FDI Packages*~~

~~NOTE IEC 62769-4 is technically identical to FDI-2024.~~

~~IEC 62769-5, *Field Device Integration (FDI) – Part 5: FDI Information Model*~~

~~NOTE 1 IEC 62769-5 is technically identical to FDI-2025.~~

~~NOTE 2 IEC 62769-5 is technically identical to FDI-2027.~~

ISO/IEC 19505-1, *Information technology – Object Management Group Unified Modeling Language (OMG UML) – Part 1: Infrastructure*

ISO/IEC 29500, (all parts) *Information technology – Document description and processing languages – Office Open XML File Formats*

3 Terms, definitions, abbreviated terms, ~~acronyms~~ symbols and conventions

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in IEC 62769-1 as well as the following apply.

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

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

3.1.1
Application Domain
isolated environment where applications execute

~~**3.1.2**~~
~~**Assembly**~~
~~reusable, version information providing, and self-describing building block of a CLR application~~

~~Note 1 to entry: This note applies to the French language only.~~

3.1.2
FDI Type Library
assembly that contains the interfaces and data types that are used for the data exchange and interaction between a UIP and an FDI Client

~~Note 1 to entry: This note applies to the French language only.~~

~~Note 2 to entry: This note applies to the French language only.~~

3.1.3
Global Assembly Cache
machine-wide code cache that stores Assemblies specifically designated to be shared by several applications

3.1.4
Windows Registry
system-defined database in which applications and system components store and retrieve configuration data

3.2 Abbreviated terms ~~and acronyms~~

For the purposes of this document, the abbreviated terms ~~and acronyms~~ given in IEC 62769-1 as well as the following apply.

CLR	Common Language Run-time
MSI	Microsoft Installer
WPF	Windows Presentation Foundation
UML	Unified Modeling Language

3.3 Symbols

Figures in this document use graphical symbols in accordance with ISO/IEC 19505-1 (UML 2.0).

3.4 Conventions

For the purposes of this document, the conventions given in IEC 62769-1 apply.

The description of Non-blocking service execution in 4.8.2 uses italics to identify a generic operation name the internal function is being applied to.

4 Technical concepts

4.1 General

4.1.1 Overview

In 4.1.2, 4.2, 4.3, 4.4, and 4.5, this document describes first the technology base for UIP implementation, the hardware and software environment including the related implementation rules. Clause 4 follows a life-cycle (use case) oriented approach.

Subclause 4.6 describes the copy deployment procedures and related implementation rules for the UIP and the FDI Client. UIP executable instantiation and termination is described in 4.7. Subclause 4.8 defines the rules about interaction between the FDI Client and the UIP. Security related definitions are written in 4.9. The service interface definitions for the FDI Client and the UIP are found in Clause 5.

4.1.2 Platforms

The UIP and FDI Client shall be built upon the Microsoft .NET Framework and executed in the .NET Common Language Run-time.

The minimum set of workstation-supported I/O devices is: mouse, keyboard, and color screen resolution of 1024 × 768 pixels.

The following Table 1 lists all the technologies and their editions that are consistent with FDI components.

Table 1 – Technology edition reference

Technology	Standard	Edition
.NET	N/A	CLR4 for UIP Implementation
EDDL	IEC 61804	~~2014~~2016
OPC UA (Parts 1-8)	IEC 62541	2015 ~~(to be published)~~
Open Packaging Convention	ISO/IEC 29500	~~2011~~2016
Extensible Markup Language (XML)	N/A	W3C, 1.0 (fifth edition)

4.1.3 FDI Type Library

The Device Access Services and the UIP Services can be modelled as .NET interfaces passing .NET data type arguments. These interfaces and data types are used for the data exchange and interaction between the UIP and the FDI Client. For runtime error handling purposes during interface method calls, .NET exceptions classes are defined.

~~The FDI .NET interfaces, data types, and exception classes are defined in a single FDI Type Library. The FDI Type Library is a strong named Assembly. The FDI Type Library is signed with a single unique key. The FDI Type Library shall be installed as part of the FDI Client installation and not with a UIP.~~

~~FDI Type Libraries shall not be registered within the Global Assembly Cache.~~

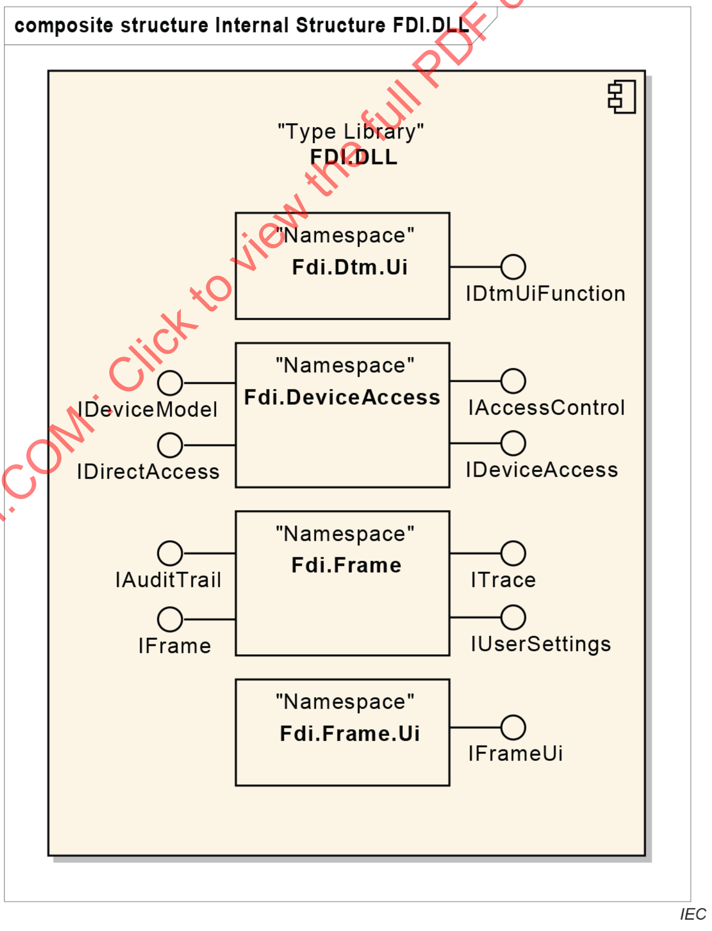
~~The FDI Client shall install FDI Library Versions for all Technology Versions that it supports.~~

~~The FDI Type Library shall be installed in such way that it is shared between the UIP and the FDI Client.~~

The FDI .NET interfaces, data types, and exception classes are defined in a single FDI Type Library. The FDI Type Library is a strong-named Assembly. The file name of this Assembly shall be 'fdi.dll'. The fdi.dll shall be versioned as per IEC 62769-1:2020, 8.1. The FDI Type Library is part of the FDI Core Technology as per IEC 62769-1:2020, 8.3.2.1 and therefore directly influences the FDI Technology Version. All Compatible changes of the fdi.dll lead to an increase of the minor portion of the FDI Technology Version. Incompatible changes lead to an increase of the major portion of the FDI Technology Version (see IEC 62769-1:2020, 8.3.2.2).

The FDI Type Library is signed with a single unique key by the issuer of the file. The FDI Type Library shall be installed separately as part of every FDI Client installation. User Interface Plug-Ins (UIP) and the FDI Client Application shall use this instance of the fdi.dll. UIPs shall not carry or deploy the FDI Type Library. The FDI Client is responsible to provide means to allow updates of this type library over time.

Figure 1 shows the FDI Type Library structure.



NOTE The composite structure diagram shows only the core interfaces that implement the interfaces defined in IEC 62769-2.

Figure 1 – FDI Type Library structure

4.2 UIP representation

The UIP Variant can contain either a single or multiple runtime modules (.NET Assembly) and their related supplementary files (for example: resource files). The runtime module of the ~~IP~~ UIP Variant is called "UIP executable". The supplementary file(s) of the UIP Variant is/are called "UIP supplement(s)".

UIP supplement(s) is/are stored under (a) subfolder(s) of the UIP executable installation directory.

EXAMPLE ~~Examples of UIP supplementary data files include~~ Resource files and application configuration data.

The RuntimeId of a UIP Variant shall be ".NET Framework CLR4", see IEC 62769-4. FDI Clients supporting this RuntimeId shall support the .NET Framework 4.6.1 or higher using the CLR4 and UIPs with this RuntimeId shall use the .NET Framework 4.6.1 or lower supporting the CLR4 (meaning .NET Framework 4.0 up to .NET Framework 4.6.1).

The UIP Variant shall be self-contained. All UIP required libraries (.NET Assemblies) required by a UIP Variant are stored within the same Folder.

4.3 UIP executable representation

The implementation of the UIP depends on the type of user interface elements that can be embedded into the user interface hosting environment of the FDI Client. UIP shall be implemented as a .NET `System.Windows.Forms` class `UserControl` or a Windows Presentation Foundation (WPF) `System.Windows.Controls` class `UserControl`.

UIP executables and their required libraries shall have strong names. The signing of a strong-named Assembly can be done using a self-generated key.

NOTE The identity of strong-named Assemblies consists of a name, version, culture, public key token and digital signature.

UIP executables and their required libraries shall be shipped with file containing the public key in order to enable Assembly verification.

4.4 UIP executable compatibility rules

~~The UIP component provided version information consists of:~~

~~<Major>.<Minor>.<Build Number>.<Revision>~~

~~UIP components using the same identity (UipId/IEC 62769-5) that are showing a different value in position <Major> are not compatible with each other. Any other difference showed in the version information between the same UIP component identities means that those UIP component identities are compatible. A newer UIP component is allowed to overwrite an older UIP component without breaking the intended functionality.~~

The compatibility rules for different versions of the UIP component are specified in IEC 62769-4.

The compilation target platform for the UIP shall be "anyCPU". If this is not feasible, the UIP shall be shipped in two variants. One UIP variant shall be compiled for target platform "x86". The second UIP variant shall be compiled for target platform "x64". The compilation platform target shall be described in the catalog.xml file, which is defined in IEC 62769-4. This catalog.xml file contains an xml element "CpuInformation" that describes the User Interface Plug-in variant. The allowed values that shall be used in the xml element "CpuInformation" are "anyCPU", "x86" or "x64".

4.5 Allowed .NET Common Language Run-time versions

4.5.1 General

Specific CLR (Common Language Run-time) versions are released for the execution of software components built with specific .NET Framework versions. The .NET CLR version 4.0 is used to execute software components built with .NET Framework 4.0. .NET Components are built for one CLR version only but can be capable to run also under a newer CLR version.

FDI Clients can be built based on CLR version 4.0 or future versions. An FDI Client has to realize the following situations when starting a UIP.

- When the UIP to be started was built for the same run-time, the UIP can be started by the FDI Client as usual.
- When the UIP to be started was built with another CLR version and is not compiled for the current running CLR version, the FDI Client shall start the UIP in a surrogate process with the adequate CLR version. (More details are described in 4.5.2.)

Taking this behavior in account, a UIP shall be developed for CLR version 4.0 or any future version. If the CLR versions do not match, the UIP shall be started in a separate process. The UIP will then not be displayed as an integrated module within the FDI Client. It is up to the FDI Client to realize the surrogate process.

4.5.2 CLR compatibility strategy

In the future, FDI Clients and UIPs will be permitted to be built on different incompatible versions of the CLR.

If an FDI Client detects that a UIP requires a CLR that is not compatible with the FDI Client, the FDI Client can use a proxy class that enables interaction with the UIP built using a different version of the CLR.

The FDI Client loads a proxy UIP executable, creates an instance of the proxy class, and delegates the execution of the UIP to this proxy. The proxy starts a process with the required CLR and executes the UIP in this surrogate process. The proxy classes provide the standard FDI interfaces. The FDI Client can use these interfaces to interact with the UIP executed in the surrogate process.

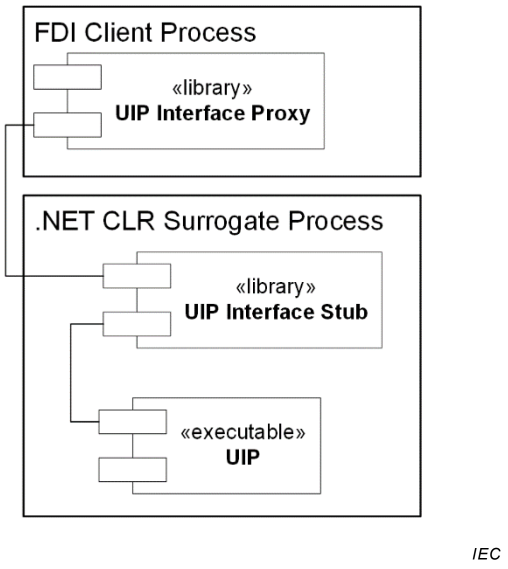


Figure 2 – .NET surrogate process

4.5.3 How to identify the .NET target platform of a UIP

The .NET target platform CLR version information for which a certain Assembly is compiled can be extracted by means of .NET Framework library functions (see Figure 3).

```
clrVersion = Assembly.LoadFrom(<Assembly Path>).ImageRuntimeVersion;
```

IEC

Figure 3 – Identification of Run-time Version

NOTE The Visual Studio[1] 2008 and 2010 IDE allow developers to select the .NET Framework target. The selection of a .NET Framework target older than the base for the current Visual Studio IDE automatically creates a configuration file listed as "app.config" within the solution explorer. This file only reflects the current complier setting. The compiler does not read that file.

4.6 ~~Installing~~ UIP Deployment

~~The FDI Server imports the UIP from an FDI Package.~~

~~The UIP installation is done per file copy only. The UIP executable shall not be registered within the Global Assembly Cache. The UIP is installed within a folder structure, which is called the UIP folder structure. The FDI Client shall manage the UIP folder structure. The UIP folder structure shall separate the UIP Variants from each other in order to avoid file name conflicts. UIP executables shall be installed to a path that allows browse read and write access.~~

~~Since the FDI Client manages the folder structure the UIP shall not perform any access to an absolute path. Any file access shall be done relative to the installation root of the UIP.~~

~~According the version management described in IEC 62769-4, the coexistence of major version changes of UIP of the same type shall be supported. This shall be done by installing a newer UIP into a separate folder. The “strong-name” rule ensures that related Assemblies can coexist during runtime.~~

The general UIP installation rules are outlined in IEC 62769-2. The UIP executable shall not be registered within the Global Assembly Cache.

The "strong-name" rule ensures that related Assemblies of different versions of the UIP can coexist during runtime.

The FDI Client implementation ensures that UIP deployment works independently from current user credentials. See the NOTE below.

NOTE Certain operating system managed folders require specific access rights, for example, modifications in folder "Program Files" require "Administrator" rights. The Windows operating system provides several means to allow an application running with restricted user rights to execute actions with administrator privileges transparent to the user, for example, special restriction handling for identified directories, services with administration rights, executables that are configured to automatically run with administration rights. The alternative is to copy UIP executables into folders writeable for "normal" users.

1 ~~Visual Studio is the trade name of Microsoft Corporation. This information is given for the convenience of users of this part of IEC 62769 and does not constitute an endorsement by IEC of the trademark holder or any of its products. Compliance does not require use of the trade name. Use of the trade name requires permission of the trade name holder.~~ Visual Studio is an example of a suitable product available commercially. This information is given for the convenience of users of this document and does not constitute an endorsement by IEC of this product.

4.7 UIP Lifecycle

4.7.1 General

The UIP state machine, outlined in IEC 62769-4, is composed of the Loaded, Created, Operational, Deactivated and Disposed states. The mechanisms affecting state changes are described in 4.7.

After the FDI Client has stored the UIP executable on the FDI Client, the FDI Client loads the UIP Assemblies dynamically into the memory and executes the related logic by calling the corresponding FDI-specified interface functions.

Subclause 4.7 describes rules about how the FDI Client shall activate and deactivate the UIP.

4.7.2 UIP Assembly activation steps

4.7.2.1 Load

The FDI Client shall load the UIP executables by using the LoadFrom mechanism. The .NET framework provides System.Reflection.Assembly.LoadFrom for this purpose:

The LoadFrom mechanism behaves as follows.

- LoadFrom loads the Assembly addressed with the file path and also the referenced Assemblies located within same directory. The argument string assemblyFile shall contain the file name of the UIP executable. The file name of the UIP executable represents the StartElementName described in IEC 62769-4.
- If an Assembly is loaded with LoadFrom, and later an Assembly in the "load context" attempts to load the same Assembly by display name, then this load attempt fails.
- If an Assembly with the same identity is already loaded (for example, by another UIP), then LoadFrom returns the Assembly that has been loaded before, even if a different file path was specified. Even a different file name does not matter. Only the identity of the Assembly is relevant.
- If an Assembly is loaded with LoadFrom, and the probing path includes an Assembly with the same identity (for example, in the Global Assembly Cache or an application directory), then this Assembly is loaded, even if a different file path was specified.
- `LoadFrom` requires the permissions `FileIOPermissionAccess.Read` and `FileIOPermissionAccess.PathDiscovery`, or `WebPermission`, on the specified path.
- `LoadFrom` loads the assembly into the default Application Domain.
- If a native Assembly image (generated by ngen.exe) exists for the specified file path, then it is not used. The Assembly cannot be loaded as domain neutral, i.e. the Assembly cannot be shared between Application Domains.

This behavior enforces deployment rules as follows.

- Rules regarding Assembly dependencies (see 4.7.2.4.2).

The FDI Client shall only use `LoadFrom`. The use of other .NET Assembly loading/object creation means is not allowed.

- Rules regarding shared Assemblies (see 4.7.2.4.3).
- A pre-compiled processor-specific machine code cannot be used.
- The security aspects regarding loading and execution of Assemblies are described in 4.9.

4.7.2.2 Create

Creating an instance of the UIP Assembly works using the .net library functions `System.Reflection.Assembly.GetTypes` and `System.Activator.CreateInstance`. The FDI type library declares a "custom attribute" named `UIPActivationClass`. This attribute shall only be added to the object implementing the interface `IDtmUiFunction` that actually implements the UIP start-up function. The attribute `UIPActivationClass` shall be used once only.

The FDI Client can now use `System.Reflection` services to clearly determine the UIP implemented activation procedure.

NOTE 1 Function System.Reflection.Assembly.GetTypes can be used to query the interface IDtmUiFunction.

NOTE 2 Function System.Attribute.GetCustomAttributes can be used for reading the additional custom attributes.

NOTE 3 The result of function invocation System.Activator.CreateInstance is an object of type IDtmUiFunction.

A data-type cast is needed.

4.7.2.3 Activate

Invocation of function `IDtmUiFunction.Init` finally activates the UIP for the user.

4.7.2.4 External libraries

4.7.2.4.1 General

UIP Assemblies can depend on external libraries (third-party libraries) and other Assemblies, for example, specific user control libraries. FDI Clients do not perform installation of UIPs, rather they dynamically load and execute the UIP. To support this usage, as well as the requirement to prevent possible problems of conflicting Assemblies, rules are specified for external libraries.

External libraries shall:

- be contained within the FDI Package;
- not require Microsoft Installer (MSI) installation;
- not require entries in the Windows Registry or the Global Assembly Cache;
- adhere to the access restrictions described in 4.9.2;
- be compatible with the platforms described in 4.1.2.

4.7.2.4.2 Loading of external libraries

The FDI Client loads the UIP Assembly, containing the UIP main class implementing interface `IDtmUiFunction`, by invocation of the .NET framework function `LoadFrom`. Referenced Assemblies that are stored in the same directory are automatically loaded together with this .NET Assembly. Referenced Assemblies that are stored in other locations (for example, in a sub-directory) have to be loaded explicitly by the UIP itself.

The UIP shall load such Assemblies also by invocation of the .NET framework function `LoadFrom`. Loading Assemblies with other .NET framework methods is not allowed.

Usage of external libraries shall not break the self-containment requirement for FDI Packages; all external libraries shall be included in the FDI UIP Package.

4.7.2.4.3 Loading of shared external libraries

An external library is a shared external library if a related .NET Assembly identity can be used from different UIP executables. The identity of a .NET Assembly matters. Installation path and Assembly filename are not relevant.

Usage of shared libraries shall not break the self-containment requirement for FDI Packages. Each of the delivered FDI Packages shall be shipped with all required UIP related libraries. The sharing mechanism comes from the .NET framework implemented optimization mechanism.

If a shared Assembly is used, then the following rules apply.

- Any incompatible change to the shared Assembly shall lead to a new identity, for example, different version number.
- Shared Assemblies shall not presume to be loaded from a specific installation path, for example, rely on the fact that some files are stored in the same directory or in a sub-directory.
- Static variables in shared Assemblies are also shared if the Assembly is loaded into the same Application Domain. Thus, static variables shall not have side effects in such scenarios. External shared libraries shall not declare static variables.
- Because of the self-containment rule defined for the FDI Package, shared Assemblies shall be deployed with all FDI Packages using a shared Assembly.

4.7.2.5 UIP Constructor invocation

Constructor and destructor implementation shall not throw exceptions. The constructor logic shall be limited to instantiate the object in terms of the internal data structure. The destructor logic shall be limited to destroy the object in terms of releasing memory resources. The constructor and the destructor shall not:

- invoke any call-back to the FDI Client,
- invoke any user interaction.

4.7.3 UIP Assembly deactivation steps

4.7.3.1 Deactivate

For UIP deactivation the FDI Client shall call the interface `IDtmUiFunction.BeginClose` and `IDtmUiFunction.EndClose`. On successful execution the UIP shall release all resources and the FDI Client shall delete all references to the UIP instance. The .NET garbage collector finally disposes the UIP runtime object.

4.7.3.2 Dispose

A .NET Assembly that is loaded into a process respectively into the related `ApplicationDomain` is never unloaded, except if the `ApplicationDomain` itself is destroyed. That means if the FDI Client loads a UIP Assembly into the default `ApplicationDomain`, then these Assemblies and all dependent Assemblies are never unloaded unless the application is closed.

The UIP Assemblies shall be developed with this .NET framework behavior in mind. To reduce the memory consumption, the following rules apply.

- Minimize the use of static variables, because these increase the memory consumption of the Assembly.
- Move UIP functionality that is not always (or rarely) needed to separate Assemblies. These Assemblies are then only automatically or manually loaded when the corresponding code is executed.

- Use shared Assemblies whenever possible.
- The FDI Client can execute .NET Assemblies in a separate Application Domain in order to have the ability to unload them.

4.8 Interaction between an FDI Client and a UIP

4.8.1 Handling of standard UI elements

UIPs shall delegate the presentation and handling of standard UI elements to the FDI Client. The standard UI elements are:

- UI Actions with standardized semantics (Apply/Close/Online Help), and
- UIP Specific status information.

To ensure a consistent user interface interaction across UIPs from different vendors, a UIP may delegate presentation and handling of additional UIP specific actions to the FDI Client. Nonetheless, UIPs are allowed to implement non-standard UI actions within their own UI area.

The set of standard UI actions and their respective semantics is fixed. However, the availability of these actions may change at any time depending on the internal state of the UIP. The set of additional UIP specific actions and their individual availability is not fixed. A UIP may add, remove, rename, enable or disable the UIP specific actions at any time depending on its requirements. The UIP has to inform the FDI Client whenever the availability of its standard actions or UIP specific actions changes (see events `IStandardActions.StandardActionItemSetChanged` and `IApplicationSpecificActions. ApplicationSpecificActionItemSetChanged)`.

An FDI Client may use dedicated UI elements, e.g. button controls, to provide direct access to the standard actions, as well as indirectly invoke them in the context of user interaction with other FDI Client UI elements. FDI Client shall always show all custom actions exposed by a UIP with dedicated UI elements.

4.8.2 Non-blocking service execution

4.8.2.1 FDI Client internal functions

The implementation of function Begin*OperationName* shall copy the content of Argument `asyncState` into member `AsyncState` of the returned `IAsyncResult` object.

The productive (time-consuming) part of the function named *OperationName* shall be performed in a different thread. The synchronization with the calling thread is handled via the `AsyncWaitHandle` object (class `WaitHandle`), which is also a member of the `IAsyncResult` object.

When processing of the productive part of the function named *OperationName* has finished, the `IAsyncResult` objects attribute `IsCompleted` shall be set to `True`. If the `AsyncCallBack` argument value is valid (not equal NULL), the FDI Client notifies the UIP using the callback.

The implementation of Cancel*OperationName* uses the argument `IAsyncResult` to identify the service that has been started with Begin*OperationName*. If Begin*OperationName* started an OPCUA service, the FDI Client shall call the OPCUA defined Cancel service.

4.8.2.2 UIP internal functions

The management of multiple asynchronous services in parallel shall be managed using the `AsyncState` object.

The `IAsyncResult` object returned by Begin*OperationName* contains the `WaitHandle` object. The UIP shall perform its own thread synchronization using the `WaitHandle` object.

4.8.2.3 Non-blocking service execution sequence

The following shows the interaction sequence between the FDI Client and the UIP. The thread management mechanisms implemented inside the FDI Client are not shown. The Interaction between an FDI Client and an FDI Server is based on Request/Response pattern. The FDI Client service request matches with the Begin*OperationName*. The AsyncCallback invocation matches with receiving the Client service response. End*OperationName* conveys the response contained results. Implementation of the non-blocking service execution does not require any thread management inside the FDI Client. Figure 4 shows an example of an IAsyncPattern based Asynchronous service execution.

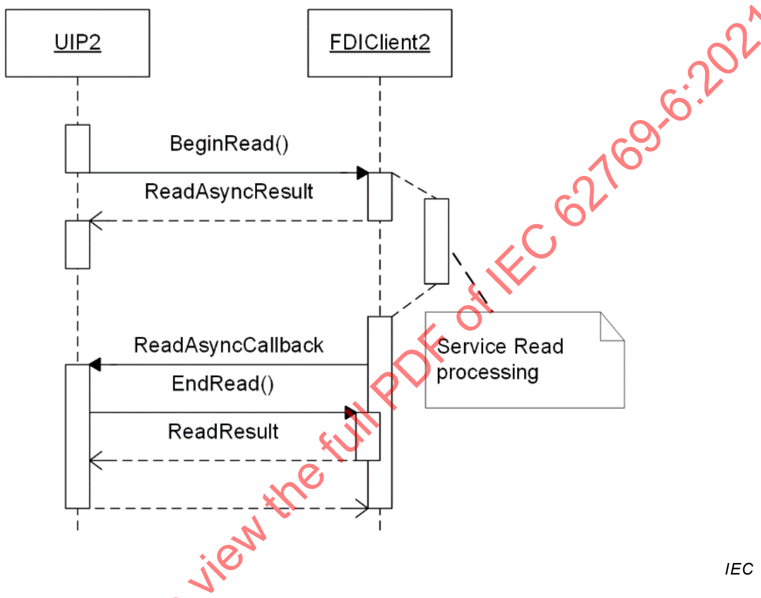
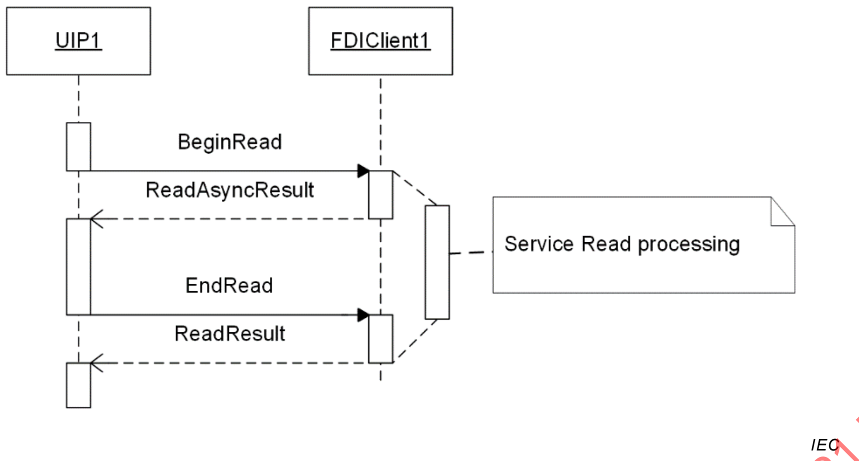


Figure 4 – IAsyncPattern based asynchronous service execution example

4.8.3 Blocking service execution

The FDI Client-provided interfaces allow performing synchronous Information Model access by using the functionality described in 4.8.1 in a way shown in Figure 5.



`ReadAsyncResult` is the object implementing the interface `IAsyncResult`.

Figure 5 – Blocking service execution example using IAsyncResult based pattern

4.8.4 Cancel service execution

Some services specified for the interface `IDeviceModel` (see Table 3) support canceling a started service by means of the function Cancel*OperationName*. The following Figure 6 will illustrate the processing sequence based on the Read service example.

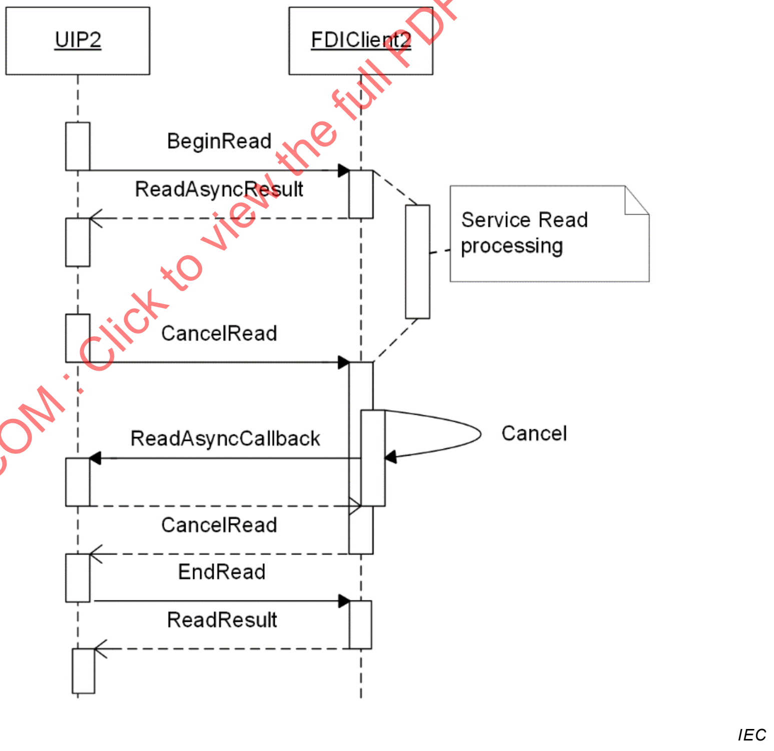


Figure 6 – Cancel service processing sequence example

The invocation of `CancelRead` triggers the FDI Client internal functions needed to cancel the active read operation. The FDI Client may not be able to cancel the operation immediately, but it should do so as soon as possible. Once the operation has been cancelled, the FDI Client notifies the UIP through the `ReadAsyncCallback`. The UIP shall then call the `EndRead` function.

NOTE A general challenge implementing this pattern is to handle race conditions properly on both sides (UIP2 and FDIClient2). If the FDI Client has forwarded the service execution via an OPC UA service, the actual service execution will run inside the FDI Server.

Depending on how the UIP is hosted, there may be three independently working processes. Therefore, the cancel request (sent by the UIP) may appear right after the FDI Server has already finished the service request. The related response sent by the FDI Server may have arrived at the FDI Client (or not). The FDI Client may invoke the ReadAsyncCallBack while the UIP invokes the CancelRead.

`ReadAsyncResult` is the object implementing the interface `IAsyncResult`.

4.8.5 Threading

4.8.5.1 Implementation rules

The UIP shall be able to receive calls in any thread.

The UIP shall not block the calls coming from the FDI Client.

The UIP shall not use the FDI Client thread to signal back the callback to the FDI Client itself. This is to prevent deadlocks and endless loops.

The UIP shall not run synchronous operations as described in 4.8.3 in the user interface thread: the user interface thread of a process shall be dedicated to receiving user inputs and perform drawing tasks only.

The UIP and FDI Client shall not block the user interface thread. The user interface shall always stay responsive. The user interface thread is shared between the different FDI user interface related objects for user input and drawing operations. If one object blocks this thread in order to perform some processing, this would affect the responsiveness of other user interfaces.

The UIP and FDI Client shall not block a Begin*OperationName* method call: a Begin*OperationName* method shall only start an asynchronous operation. The caller shall not be blocked.

4.8.6 Timeout

The interfaces referred in Clause 5 enable asynchronous service execution. The time for the execution of such services depends on performance constraints related to: bus communication, FDI Client/FDI Server performance. The rules listed below target the system interoperability regarding the prevention of "Race Conditions". The general rule is that the component is allowed to manage timeout handling only for those processes that are completely under the control of that component. The following list shows which elements of the entire system are allowed to implement the timeout detection function.

- UIP: The UIP shall not implement timeout detection.
- Business Logic: The Business Logic shall not implement timeout detection (FDI Package).
- FDI Client: The FDI Client shall implement timeout detection. In the case of OPC UA, the related support is built into the OPC UA communication stacks. Timeout detected during operations performed on behalf of the UIP shall be forwarded as negative function result codes.
- FDI Server: The FDI Server shall implement timeout detection. In the case of OPC UA, the related support is built into the OPC UA communication stacks.
- Communication Server: The Communication Server implements timeout detection for the OPC UA connection according to the OPC UA Specification. Related support is built into the OPC UA communication stacks. Additionally, the Communication Server implements timeout detection limited to the network directly connected to the physical port connected to the Communication Server.

4.8.7 Exception handling

An important specification goal is to make a clear distinction between software quality problems and anticipated processing states. Therefore, the specification defines two general exception categories:

a) Exceptions that indicate software states or events that have not been anticipated during the software development are considered as software quality issues (Run-time error).

b) Exceptions indicating anticipated software operation failures.

Examples of software quality issues indicated by exceptions are:

- function argument type mismatch;
- function argument value range mismatch;
- division by zero;
- NULL Pointer reference.

Examples of anticipated error handling are:

- communication problem handling;
- general IO data processing;
- user input errors.

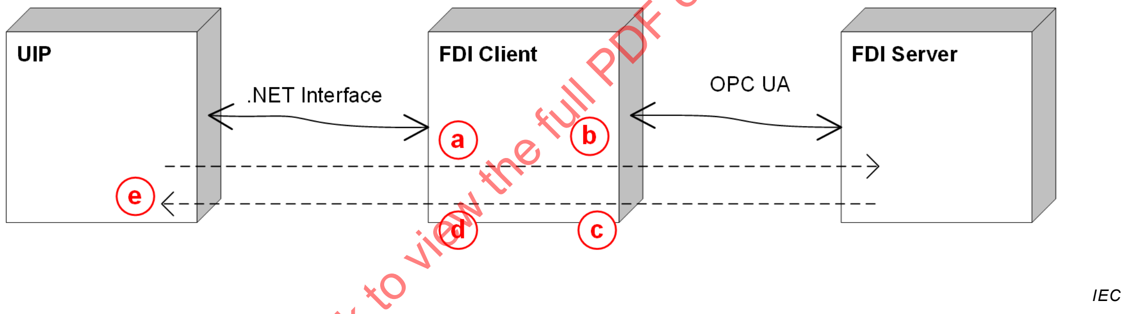


Figure 7 – Exception source

According to the FDI Architecture, exceptions can occur in different steps of the service processing, see Figure 7:

a) passing the request from the UIP to the FDI Client:

b) request forwarding inside the FDI Client;

c) processing the response from the FDI Server;

d) forwarding the response to the UIP;

e) response processing inside the UIP.

Service processing problems detected inside the FDI Server and beyond are handled through OPC UA defined service results.

Regarding the implementation of the `IAsyncResult` pattern, the following rules apply.

– Any failure occurring with step a) shall be reported by an exception thrown by the Begin*OperationName*.

– Any failure occurring during steps b) to e) shall be handled by the corresponding component. The execution of the End*OperationName* shall then report the failure via an exception.

4.8.8 Type safe interfaces

The Information Model hosts device variables of different types. The values of such variables are transferred using the class DataValue (FDI Interfaces and Data Types.CHM).

The Device Access Services support writing or reading multiple variables within one service. The data type chosen for data transport is `DataValue` implementing the type safe transport because of the `DataValue` property `Datatype` describing the value data type by means of `Datatype` enumeration. Because the `DataValue` property Value get/set functions use data type Object to convey the actual value, the data receiver (UIP) shall verify the data type before data processing.

4.8.9 Globalization and localization

~~The default locale for UIP is English/(US).~~

~~Optional language support is allowed according to market needs.~~

UIP localization support can be implemented through resource files (.res(x)) or satellite Assemblies.

The FDI Client shall set the locale and country information that shall be used by the UIP by means of the arguments `currentRegion` and `currentCulture` that are submitted with the invocation of method `Fdi.Dtm.Ui.IDtmUiFunction.Init`. The UIP shall not derive locale information via the `Thread.CurrentUICulture`. The data type for `currentRegion` is `RegionInfo` defined in the .NET namespace `System.Globalization`. The data type for `currentCulture` is `CultureInfo` defined in the .NET namespace `System.Globalization`.

4.8.10 WPF Control handling

If a UIP implementation is based on WPF `UserControl`, the UIP inherits the interface from the class `UserControl`, which means there will be more methods attributes and events available for the FDI Client that are not covered by the FDI specification. Conversely, a UIP implements the accessibility function. The related rules affect, on the one hand ~~touch~~, the quality of the UIP product and, on the other hand, the interoperability.

4.8.11 Win Form handling

If a UIP implementation is based on Windows Forms, the UIP inherits from the class `UserControl`, which means there will be more methods attributes and events available for the FDI Client that are not covered by the FDI specification. Conversely, a UIP implements the accessibility function. The related rules affect, on the one hand ~~touch~~, the quality of the UIP product and, on the other hand, the interoperability. The FDI Client shall make thread-safe calls to the `Windows.Forms` controls.

4.9 Security

4.9.1 General

The goal of security is to protect a system against threats compromising the system's stability, integrity and sensitive data.

System-wide security begins with the design process, which is out of the scope of standardization. From the system's perspective, security is about controlling access to resources, such as application components, data, and hardware. The .NET framework provides support for constraining access to resources. The system security is based on control over access permissions.

A different approach is based on certification and authentication. Since ~~any FDI Package needs~~ all FDI Packages need compliance testing and certification, the presumption is that such certified FDI Packages don't pose any threats to a system. This means that a UIP could be executed with fully trusted permissions.

While an over-constrained system could lead into functional problems ~~an un-constrained~~, unconstrained permissions can be seen as security threats. Subclause 4.9 represents a compromise between both ways.

4.9.2 Access permissions

4.9.2.1 General

~~The access permissions for a UIP are enforced by the .net CLR run-time system following the security policy. The security policy is a configurable set of rules defining the constraints for resource access. Only administrators shall modify or customize the security policy according to the specific needs of their organizations. The CLR runtime grants permissions to both Assemblies and Application Domains based on the security policy.~~

~~Identity permissions represent characteristics that identify an Assembly. The CLR grants identity permissions to an Assembly based on the information it obtains about the Assembly.~~

Within 4.9.2, technology-specific permissions and restrictions are specified in addition to the one specified in IEC 62769-2. The permissions and restrictions shall be enforced by the FDI Client. The implementation rules define how this shall be implemented.

4.9.2.2 Technology specific UIP permissions and restrictions

~~The UIP access permissions defined in this part of IEC 62769 are specified according to use cases that need to be implemented with UIP as follows.~~

a) ~~Reference data bases and help files (UIP Supplementary data) shall be provided with UIP and stored within in the UIP installation folder on the FDI Client. The access permission to this folder is read-only.~~
b) ~~UIP specific data like Valve Signatures shall be stored in the Information Model and accessed by means of the EDD element.~~
c) ~~The export/import use case shall be supported by allowing the user to save and load data from a user specified folder. The access permissions to this folder are defined by means operating system administrated user credentials. The export/import function enables data migration, backup/restore.~~
d) ~~User settings, preferences or data caching shall be done via the services SaveUserSettings and LoadUserSettings specified in IEC 62769-2.~~
e) ~~Launching of an Active-X component is not allowed.~~
f) ~~Sharing of UIP specific data can be done either through the Information Model or by means of the export import function described in c).~~
g) ~~If a printer is available, access to that printer is allowed.~~
h) ~~A UIP executable shall not implement role based access permission constraints.~~
i) ~~A UIP shall not access the operating system registry.~~

~~The FDI Client shall grant the following list of minimum set of permissions:~~

a) ~~A UIP executable is allowed to read UIP supplementary data files from the installation directory and below (see 4.2). The UIP is not allowed to browse the file system above its installation root.~~
b) ~~A UIP executable shall not perform internet access.~~
c) ~~A UIP executable shall not perform local area network (LAN) access.~~

The general UIP permissions and restrictions are specified in IEC 62769-2. The permissions and restrictions specified in 4.9.2 are specific to .NET-based UIPs.

a) Launching of an Active-X component is not allowed.

b) A UIP shall not write to the operating system registry. Read access to registry is allowed.

The FDI Client shall restrict the UIP permissions in accordance with this list.

4.9.2.3 Implementation rules

~~Sandboxing enables running the code in an environment with restricted permissions. An FDI Client shall limit the access permissions given to a UIP.~~

~~The FDI Client can run the UIP within an Application Domain providing a sandbox for the UIP. The Application Domain is used for running the partially trusted UIP with permissions that define the availability of protected resources when running within that Application Domain. The UIP that runs inside the Application Domain is bound by the permissions associated with the Application Domain and is allowed to access only the specified resources.~~

~~The FDI Client shall use the function `System.AppDomain.CreateDomain(String, Evidence, AppDomainSetup, PermissionSet, StrongName[])` method overload to specify the permission set for the UIP that runs in a sandbox. This overload enables the FDI Client to specify the exact level of code access security specified in 4.9.2.2.~~

~~NOTE Assemblies that are loaded into an Application Domain by using this overload can either have the specified grant set only, or can be fully trusted. The Assembly is granted full trust if it is in the Global Assembly Cache or listed in the fullTrustAssemblies (the StrongName) array parameter.~~

~~Only Assemblies known to be fully trusted should be added to the fullTrustAssemblies list. The list of trusted assemblies is managed by the operating system.~~

~~The PermissionSet assigned to the Application Domain running the UIP (UIP-Sandbox) shall be initialized with~~

~~PermissionSet(PermissionState.None)~~

~~The UIP permission set shall contain:~~

~~a) `FileDialogPermissionAccess`~~

~~b) `FileIOPermissionAccess`~~

~~c) `UIPermissionWindow`~~

~~d) `SecurityPermissionFlag.Execution`~~

~~e) `ReflectionPermission`~~

In order to restrict the permissions of a UIP, an FDI Client shall execute a UIP in a separate process (not the FDI Client process) that should run with a separate user account (4.5.2 describes how an FDI Client can manage hosting a UIP in a different process). By restricting the permissions of the separate user account, the FDI Client also restricts the permissions of the UIP running under the separate user account.

How the separate user account is created and configured is host specific. That includes

– whether it is a local or a domain account;

– whether it is used only for UIP hosting or also for other purposes;

– whether it is reused for all UIP processes of the same FDI Client instance or several FDI Clients;

– how restrictions are implemented (e.g. limited access to the file system by mechanism of the operating system, blocking network access by a firewall).

Whether the FDI Client executes each UIP instance in its own process, or executes several or all UIP instances in the same process, is FDI Client-specific.

As UIPs run in a separate process, UIP developers need to be aware that opening a modal dialog with operating system mechanisms will only provide modality within the Windows of that process. To avoid this behavior, UIP developers can start the modal UIP or start a new modal UIP.

4.9.3 Code identity concept

The ability to uniquely identify UIP executables contributes to the system's security.

As earlier described in 4.3, UIP executables shall be signed with strong names. Strong ~~names~~-named signed .NET Assemblies enable:

– unique identification of UIP executables;

– code integrity verification.

NOTE The benefit of strong-named UIP executables is lost if this Assembly dynamically loads other library Assemblies that are not signed with strong names.

5 Interface definition

Table 2 to Table 9 specify the mapping between the abstract services specified in IEC 62769-2 and the corresponding .NET implementation, which is found in the type library file "FDI.DLL" and a related help file "FDI Interfaces and Data Types.chm".

NOTE The Files "FDI.DLL" and "FDI Interfaces and Data Types.chm" can be obtained from the fieldbus organizations. (See also http://www.fdi-cooperation.com).

Table 2 specifies the mapping of the Base Property Services.

Table 2 – Base Property Services

Abstract Service	.NET Implementation
GetDeviceAccessInterfaceVersion	IDeviceAccess.Version
GetOnlineAccessAvailability	IDeviceAccess.OnlineAccessAvailable

Table 3 specifies the mapping of the Device Model Services.

Table 3 – Device Model Services

Abstract Service	.NET Implementation
Browse	IDeviceModel.BeginBrowse IDeviceModel.CancelBrowse IDeviceModel.EndBrowse
Read	IDeviceModel.BeginRead IDeviceModel.CancelRead IDeviceModel.EndRead
Write	IDeviceModel.BeginWrite IDeviceModel.CancelWrite IDeviceModel.EndWrite
CreateSubscription	IDeviceModel.BeginCreateSubscription IDeviceModel.EndCreateSubscription
Subscribe	IDeviceModel.BeginSubscribe IDeviceModel.EndSubscribe
Unsubscribe	IDeviceModel.BeginUnsubscribe IDeviceModel.EndUnsubscribe
DeleteSubscription	IDeviceModel.BeginDeleteSubscription IDeviceModel.EndDeleteSubscription
DataChangeCallback	DataChangeCallback

Table 4 specifies the mapping of the Access Control Services.

Table 4 – Access Control Services

Abstract Service	.NET Implementation
InitLock	IAccessControl.BeginInitLock IAccessControl.EndInitLock
ExitLock	IAccessControl.BeginExitLock IAccessControl.EndExitLock

Table 5 specifies the mapping of the Direct Access Services.

Table 5 – Direct Access Services

Abstract Service	.NET Implementation
InitDirectAccess	IDirectAccess.BeginInitDirectAccess IDirectAccess.EndInitDirectAccess
ExitDirectAccess	IDirectAccess.BeginExitDirectAccess IDirectAccess.EndExitDirectAccess
Transfer	IDirectAccess.BeginTransfer IDirectAccess.EndTransfer

Table 6 specifies the mapping of the Hosting Services.

Table 6 – Hosting Services

Abstract Service	.NET Implementation
GetClientTechnologyVersion	Fdi.Frame.IFrame.Version

Abstract Service	.NET Implementation
OpenUserInterface[a]	Fdi.Frame.Ui.IFrameUi.BeginOpenDtmUiModal Fdi.Frame.Ui.IFrameUi.EndOpenDtmUiModal Fdi.Frame.Ui.IFrameUi.BeginOpenDtmUi Fdi.Frame.Ui.IFrameUi.EndOpenDtmUi
CloseUserInterface[a]	Fdi.Dtm.Ui.CloseMeRequestHandler ~~[c]~~[b] Fdi.Frame.Ui.IFrameUi.BeginCloseDtmUi Fdi.Frame.Ui.IFrameUi.EndCloseDtmUi
LogAuditTrailMessage	Fdi.Frame.IAuditTrail.Notify
SaveUserSettings	Fdi.Frame.IUserSettings.SaveUserSettings
LoadUserSettings	Fdi.Frame.IUserSettings.LoadUserSettings
Trace	Fdi.Frame.ITrace.TraceData Fdi.Frame.ITrace.TraceEvent
ShowMessageBox	Fdi.Frame.Ui.IFrameUi.ShowMessageBox
ShowProgressBar	Fdi.Frame.Ui.IFrameUi.ShowProgress
CancelCallback	Fdi.Frame.Ui.CancelEventHandler
UpdateShowProgressBar	Fdi.Frame.Ui.IProgressUi.UpdateProgress
EndShowProgressBar	Fdi.Frame.Ui.IProgressUi.EndProgress
DefaultResult	System.Windows.MessageBoxResult
ButtonSet	System.Windows.MessageBoxButton
AcknStyle	System.Windows.MessageBoxImage
ExportFile	Fdi.Frame.ImportExport.IFileExport.BeginInitExport Fdi.Frame.ImportExport.IFileExport.EndInitExport Fdi.Frame.ImportExport.IFileExport.BeginWriteExport Fdi.Frame.ImportExport.IFileExport.EndWriteExport Fdi.Frame.ImportExport.IFileExport.BeginFinishExport Fdi.Frame.ImportExport.IFileExport.EndFinishExport
CancelExportFile	Fdi.Frame.ImportExport.IFileExport.BeginFinishExport Fdi.Frame.ImportExport.IFileExport.EndFinishExport
ImportFile	Fdi.Frame.ImportExport.IFileExport.BeginInitImport Fdi.Frame.ImportExport.IFileExport.EndInitImport Fdi.Frame.ImportExport.IFileExport.BeginReadImport Fdi.Frame.ImportExport.IFileExport.EndReadImport Fdi.Frame.ImportExport.IFileExport.BeginFinishImport Fdi.Frame.ImportExport.IFileExport.EndFinishImport
CancelImportFile	Fdi.Frame.ImportExport.IFileExport.BeginFinishImport Fdi.Frame.ImportExport.IFileExport.EndFinishImport

Abstract Service	.NET Implementation
OpenDefaultApplication	Fdi.Frame.DefaultApplication.IOpenDefaultApplication.BeginInitOpenDefaultApplication Fdi.Frame.DefaultApplication.IOpenDefaultApplication.EndInitOpenDefaultApplication Fdi.Frame.DefaultApplication.IOpenDefaultApplication.BeginWriteOpenDefaultApplication Fdi.Frame.DefaultApplication.IOpenDefaultApplication.EndWriteOpenDefaultApplication Fdi.Frame.DefaultApplication.IOpenDefaultApplication.BeginFinishOpenDefaultApplication Fdi.Frame.DefaultApplication.IOpenDefaultApplication.EndFinishOpenDefaultApplication
GetEnvironmentProperties	Fdi.Frame.HostProperty. GetHostingProperties
[a] Functions `OpenUserInterface`, `CloseUserInterface`~~, `OpenModalUserInterface`~~ shall only be started using the operation pattern described in 4.8.2.3. ~~[b] Functions are to be used to manage an additional UIP.~~ [~~c~~b] To be used by the UIP to close itself.	

Table 7 specifies the mapping of the UIP Services.

Table 7 – UIP Services

Abstract Service	.NET Implementation
Activate	Fdi.Dtm.Ui.IDtmUiFunction.Init
Deactivate	Fdi.Dtm.Ui.IDtmUiFunction.BeginClose [a] Fdi.Dtm.Ui.IDtmUiFunction.EndClose [a]
SetSystemLabel	Fdi.Dtm.Ui.IDtmUiFunction.SystemGuiLabel
SetTraceLevel	Fdi.Dtm.Ui.IDtmUiFunction.TraceLevel
TraceLevel	Fdi.Frame.TraceEventType
InvokeStandardAction~~(*1)~~	Fdi.Dtm.Ui.IStandardActions.InvokeStandardAction
InvokeApplicationSpecificAction	Fdi.Dtm.Ui. IApplicationSpecificActions.InvokeApplicationSpecificAction
GetStandardActionItems	Fdi.Dtm.Ui.IStandardActions.ActionItemSet
GetApplicationSpecificActionItems	Fdi.Dtm.Ui. IApplicationSpecificActions.ApplicationSpecificActionItemSet
StandardActionItemsChangeCallback	Fdi.Dtm.Ui.IStandardActions.StandardActionItemSetChanged
ApplicationSpecificActionItemsChangeCallback	Fdi.Dtm.Ui. IApplicationSpecificActions.ApplicationSpecificActionItemSetChanged
[a] The Deactivate service specified response `deactivateCancelled` maps to the exception `FdiCannotCloseUiException` to be thrown by the UIP if the UIP has problems with the deactivation.	

Table 8 specifies the mapping of the base data types.

Table 8 – Base Data Types

Base data type	.NET Implementation	
Boolean	Fdi.DataTypes.BooleanValue	enum DataType.Boolean
String	Fdi.DataTypes.StringValue	enum DataType.String
ByteString	Fdi.DataTypes.BinaryValue	enum DataType.Binary
UtcTime	Fdi.DataTypes.DateTimeValue	enum DataType.DateTime
Int8	Fdi.DataTypes.SByteValue	enum DataType.SByte
Int16	Fdi.DataTypes.ShortValue	enum DataType.Short
Int32	Fdi.DataTypes.IntValue	enum DataType.Int
Int64	Fdi.DataTypes.LongValue	enum DataType.Long
Byte	Fdi.DataTypes.ByteValue	enum DataType.Byte
UInt16	Fdi.DataTypes.UShortValue	enum DataType.UShort
UInt32	Fdi.DataTypes.UIntValue	enum DataType.UInt
UInt64	Fdi.DataTypes.ULongValue	enum DataType.ULong
Float	Fdi.DataTypes.FloatValue	enum DataType.Float
Double	Fdi.DataTypes.DoubleValue	enum DataType.Double
Duration	Fdi.DataTypes.TimeSpanValue	enum DataType.TimeSpan

Table 9 specifies the mapping of the special data types.

Table 9 – Special Types

Special Data type	.NET Implementation	
Attribute Ids	Fdi.DeviceAccess.AttributeType	
Variant	Fdi.DeviceAccess.DataValue	
NodeSpecifier	Fdi.DeviceAccess.NodeSpecifier	
Data Value	Fdi.DeviceAccess.ReadResult	
Localized Text	Fdi. DataTypes.LocalizedTextValue	enum DataType.LocalizedText
Range	Fdi. DataTypes.RangeValue	enum DataType.Range
EU Information	Fdi. DataTypes.EUInfoValue	enum DataType.EngineeringUnit
Enum Value	Fdi. DataTypes.EnumValue	enum DataType.Enumerator
InnerErrorInfo	Fdi.DeviceAccess.InnerErrorInfo	
NumericRange	Fdi.DeviceAccess.ArrayIndexRange	

Data arrays can be conveyed using class `Fdi.DataTypes.ArrayValue`.

Detailed interface definition and interface documentation are available in:

– FDI.DLL (.NET Assembly)

– FDI Interfaces and Data Types.CHM (Help File)

~~Bibliography~~

~~FDI-2021, *FDI Project Technical Specification – Part 1: Overview* <available at www.fdi-cooperation.com>~~

~~FDI-2022, *FDI Project Technical Specification – Part 2: FDI Client* <available at www.fdi-cooperation.com>~~

~~FDI-2023, *FDI Project Technical Specification – Part 3: FDI Server* <available at www.fdi-cooperation.com>~~

~~FDI-2024, *FDI Project Technical Specification – Part 4: FDI Packages* <available at www.fdi-cooperation.com>~~

~~FDI-2025, *FDI Project Technical Specification – Part 5: FDI Information Model* <available at www.fdi-cooperation.com>~~

~~FDI-2026, *FDI Project Technical Specification – Part 6: FDI Technology Mapping* <available at www.fdi-cooperation.com>~~

~~FDI-2027, *FDI Project Technical Specification – Part 7: FDI Communication Devices* <available at www.fdi-cooperation.com>~~

IEC 62769-6

Edition 2.0 2021-02

INTERNATIONAL STANDARD

NORME INTERNATIONALE

colour inside

Field device integration (FDI) –
Part 6: Technology Mapping

Intégration des appareils de terrain (FDI) –
Partie 6: Mapping de technologies

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

____

FIELD DEVICE INTEGRATION (FDI) –

Part 6: Technology Mapping

FOREWORD

International Standard IEC 62769-6 has been prepared by subcommittee 65E: Devices and integration in enterprise systems, of IEC technical committee 65: Industrial-process measurement, control and automation.

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

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

a) redesign of the security concept for UIP execution.

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

FDIS	Report on voting
65E/763/FDIS	65E/773/RVD

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

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

A list of all parts in the IEC 62769 series, published under the general title *Field Device Integration (FDI)*, can be found on the IEC website.

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

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

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

INTRODUCTION

The IEC 62769 series has the general title *Field Device Integration (FDI)* and the following parts:

– Part 1: Overview
– Part 2: FDI Client
– Part 3: FDI Server
– Part 4: FDI Packages
– Part 5: FDI Information Model
– Part 6: FDI Technology Mapping
– Part 7: FDI Communication Devices
– Part 100: Profiles – Generic Protocol Extensions
– Part 101-1: Profiles – Foundation Fieldbus H1
– Part 101-2: Profiles – Foundation Fieldbus HSE
– Part 103-1: Profiles – PROFIBUS
– Part 103-4: Profiles – PROFINET
– Part 109-1: Profiles – HART and WirelessHART
– Part 115-2: Profiles – Protocol-specific Definitions for Modbus RTU
– Part 150-1: Profiles – ISA 100.11a

FIELD DEVICE INTEGRATION (FDI) –

Part 6: Technology Mapping

1 Scope

This part of IEC 62769 specifies the technology mapping for the concepts described in the Field Device Integration (FDI) standard. The technology mapping focuses on implementation regarding the components FDI Client and User Interface Plug-in (UIP) that are specific only to the WORKSTATION platform/.NET as defined in IEC 62769-4.

2 Normative references

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

IEC 61804 (all parts), *Function blocks (FB) for process control and Electronic Device Description Language (EDDL)*

IEC 62769-1, *Field Device Integration (FDI) – Part 1: Overview*

IEC 62769-2, *Field Device Integration (FDI) – Part 2: FDI Client*

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

IEC 62541 (all parts), *OPC Unified Architecture*

ISO/IEC 19505-1, *Information technology – Object Management Group Unified Modeling Language (OMG UML) – Part 1: Infrastructure*

ISO/IEC 29500, (all parts) *Information technology – Document description and processing languages – Office Open XML File Formats*

3 Terms, definitions, abbreviated terms, symbols and conventions

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in IEC 62769-1 as well as the following apply.

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

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

3.1.1
Application Domain
isolated environment where applications execute

3.1.2
FDI Type Library
assembly that contains the interfaces and data types that are used for the data exchange and interaction between a UIP and an FDI Client

3.1.3
Global Assembly Cache
machine-wide code cache that stores Assemblies specifically designated to be shared by several applications

3.1.4
Windows Registry
system-defined database in which applications and system components store and retrieve configuration data

3.2 Abbreviated terms

For the purposes of this document, the abbreviated terms given in IEC 62769-1 as well as the following apply.

CLR	Common Language Run-time
MSI	Microsoft Installer
WPF	Windows Presentation Foundation
UML	Unified Modeling Language

3.3 Symbols

Figures in this document use graphical symbols in accordance with ISO/IEC 19505-1 (UML 2.0).

3.4 Conventions

For the purposes of this document, the conventions given in IEC 62769-1 apply.

The description of Non-blocking service execution in 4.8.2 uses italics to identify a generic operation name the internal function is being applied to.

4 Technical concepts

4.1 General

4.1.1 Overview

In 4.1.2, 4.2, 4.3, 4.4, and 4.5, this document describes first the technology base for UIP implementation, the hardware and software environment including the related implementation rules. Clause 4 follows a life-cycle (use case) oriented approach.

Subclause 4.6 describes the copy deployment procedures and related implementation rules for the UIP and the FDI Client. UIP executable instantiation and termination is described in 4.7. Subclause 4.8 defines the rules about interaction between the FDI Client and the UIP. Security related definitions are written in 4.9. The service interface definitions for the FDI Client and the UIP are found in Clause 5.

4.1.2 Platforms

The UIP and FDI Client shall be built upon the Microsoft .NET Framework and executed in the .NET Common Language Run-time.

The minimum set of workstation-supported I/O devices is: mouse, keyboard, and color screen resolution of 1024 × 768 pixels.

The following Table 1 lists all the technologies and their editions that are consistent with FDI components.

Table 1 – Technology edition reference

Technology	Standard	Edition
.NET	N/A	CLR4 for UIP Implementation
EDDL	IEC 61804	2016
OPC UA (Parts 1-8)	IEC 62541	2015
Open Packaging Convention	ISO/IEC 29500	2016
Extensible Markup Language (XML)	N/A	W3C, 1.0 (fifth edition)

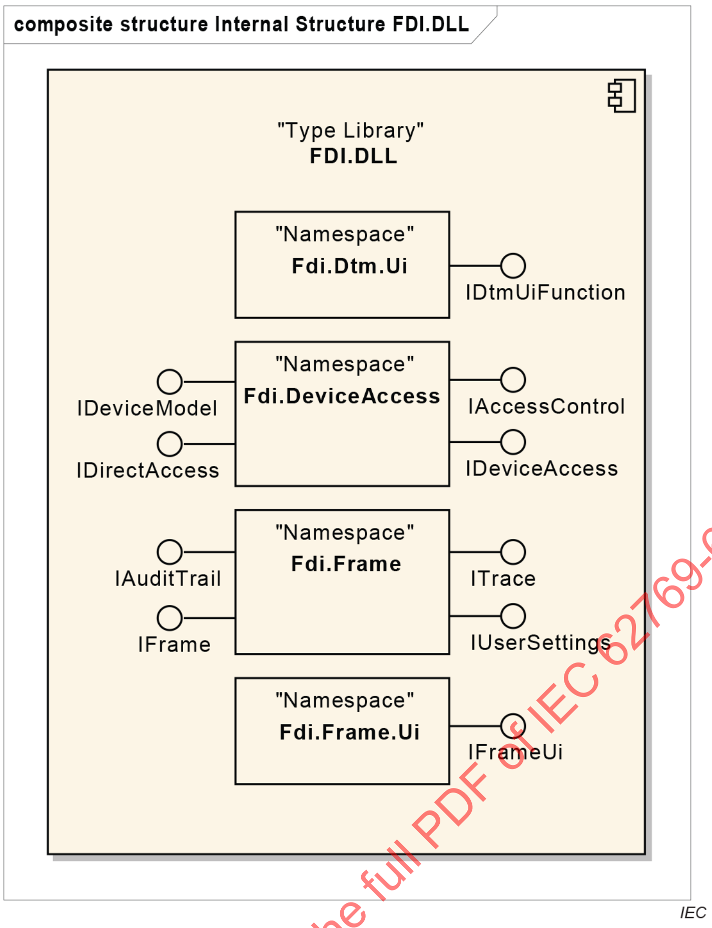
4.1.3 FDI Type Library

The Device Access Services and the UIP Services can be modelled as .NET interfaces passing .NET data type arguments. These interfaces and data types are used for the data exchange and interaction between the UIP and the FDI Client. For runtime error handling purposes during interface method calls, .NET exceptions classes are defined.

The FDI .NET interfaces, data types, and exception classes are defined in a single FDI Type Library. The FDI Type Library is a strong-named Assembly. The file name of this Assembly shall be ‘fdi.dll’. The fdi.dll shall be versioned as per IEC 62769-1:2020, 8.1. The FDI Type Library is part of the FDI Core Technology as per IEC 62769-1:2020, 8.3.2.1 and therefore directly influences the FDI Technology Version. All Compatible changes of the fdi.dll lead to an increase of the minor portion of the FDI Technology Version. Incompatible changes lead to an increase of the major portion of the FDI Technology Version (see IEC 62769-1:2020, 8.3.2.2).

The FDI Type Library is signed with a single unique key by the issuer of the file. The FDI Type Library shall be installed separately as part of every FDI Client installation. User Interface Plug-Ins (UIP) and the FDI Client Application shall use this instance of the fdi.dll. UIPs shall not carry or deploy the FDI Type Library. The FDI Client is responsible to provide means to allow updates of this type library over time.

Figure 1 shows the FDI Type Library structure.



NOTE The composite structure diagram shows only the core interfaces that implement the interfaces defined in IEC 62769-2.

Figure 1 – FDI Type Library structure

4.2 UIP representation

The UIP Variant can contain either a single or multiple runtime modules (.NET Assembly) and their related supplementary files (for example: resource files). The runtime module of the UIP Variant is called "UIP executable". The supplementary file(s) of the UIP Variant is/are called "UIP supplement(s)".

UIP supplement(s) is/are stored under (a) subfolder(s) of the UIP executable installation directory.

EXAMPLE Resource files and application configuration data.

The RuntimeId of a UIP Variant shall be ".NET Framework CLR4", see IEC 62769-4. FDI Clients supporting this RuntimeId shall support the .NET Framework 4.6.1 or higher using the CLR4 and UIPs with this RuntimeId shall use the .NET Framework 4.6.1 or lower supporting the CLR4 (meaning .NET Framework 4.0 up to .NET Framework 4.6.1).

The UIP Variant shall be self-contained. All UIP required libraries (.NET Assemblies) required by a UIP Variant are stored within the same Folder.

4.3 UIP executable representation

The implementation of the UIP depends on the type of user interface elements that can be embedded into the user interface hosting environment of the FDI Client. UIP shall be implemented as a .NET `System.Windows.Forms` class `UserControl` or a Windows Presentation Foundation (WPF) `System.Windows.Controls` class `UserControl`.

UIP executables and their required libraries shall have strong names. The signing of a strong-named Assembly can be done using a self-generated key.

NOTE The identity of strong-named Assemblies consists of a name, version, culture, public key token and digital signature.

UIP executables and their required libraries shall be shipped with file containing the public key in order to enable Assembly verification.

4.4 UIP executable compatibility rules

The compatibility rules for different versions of the UIP component are specified in IEC 62769-4.

The compilation target platform for the UIP shall be "anyCPU". If this is not feasible, the UIP shall be shipped in two variants. One UIP variant shall be compiled for target platform "x86". The second UIP variant shall be compiled for target platform "x64". The compilation platform target shall be described in the catalog.xml file, which is defined in IEC 62769-4. This catalog.xml file contains an xml element "CpuInformation" that describes the User Interface Plug-in variant. The allowed values that shall be used in the xml element "CpuInformation" are "anyCPU", "x86" or "x64".

4.5 Allowed .NET Common Language Run-time versions

4.5.1 General

Specific CLR (Common Language Run-time) versions are released for the execution of software components built with specific .NET Framework versions. The .NET CLR version 4.0 is used to execute software components built with .NET Framework 4.0. .NET Components are built for one CLR version only but can be capable to run also under a newer CLR version.

FDI Clients can be built based on CLR version 4.0 or future versions. An FDI Client has to realize the following situations when starting a UIP.

- When the UIP to be started was built for the same run-time, the UIP can be started by the FDI Client as usual.
- When the UIP to be started was built with another CLR version and is not compiled for the current running CLR version, the FDI Client shall start the UIP in a surrogate process with the adequate CLR version. (More details are described in 4.5.2.)

Taking this behavior in account, a UIP shall be developed for CLR version 4.0 or any future version. If the CLR versions do not match, the UIP shall be started in a separate process. The UIP will then not be displayed as an integrated module within the FDI Client. It is up to the FDI Client to realize the surrogate process.

4.5.2 CLR compatibility strategy

In the future, FDI Clients and UIPs will be permitted to be built on different incompatible versions of the CLR.

If an FDI Client detects that a UIP requires a CLR that is not compatible with the FDI Client, the FDI Client can use a proxy class that enables interaction with the UIP built using a different version of the CLR.

The FDI Client loads a proxy UIP executable, creates an instance of the proxy class, and delegates the execution of the UIP to this proxy. The proxy starts a process with the required CLR and executes the UIP in this surrogate process. The proxy classes provide the standard FDI interfaces. The FDI Client can use these interfaces to interact with the UIP executed in the surrogate process.

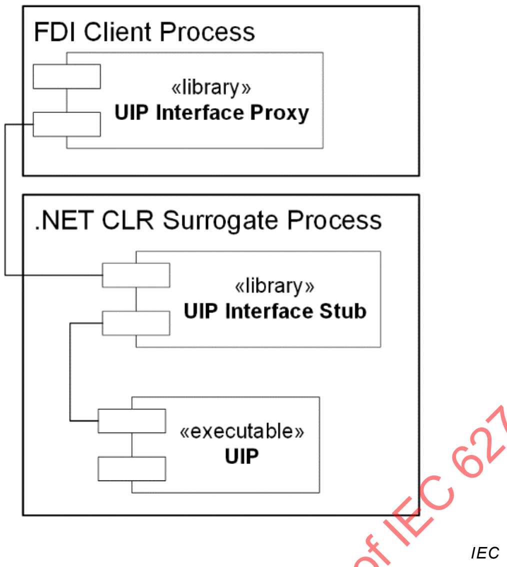


Figure 2 – .NET surrogate process

4.5.3 How to identify the .NET target platform of a UIP

The .NET target platform CLR version information for which a certain Assembly is compiled can be extracted by means of .NET Framework library functions (see Figure 3).

```
clrVersion = Assembly.LoadFrom(<Assembly Path>).ImageRuntimeVersion;
```

IEC

Figure 3 – Identification of Run-time Version

NOTE The Visual Studio[1] 2008 and 2010 IDE allow developers to select the .NET Framework target. The selection of a .NET Framework target older than the base for the current Visual Studio IDE automatically creates a configuration file listed as "app.config" within the solution explorer. This file only reflects the current complier setting. The compiler does not read that file.

4.6 UIP Deployment

The general UIP installation rules are outlined in IEC 62769-2. The UIP executable shall not be registered within the Global Assembly Cache.

The "strong-name" rule ensures that related Assemblies of different versions of the UIP can coexist during runtime.

The FDI Client implementation ensures that UIP deployment works independently from current user credentials. See the NOTE below.

[1] Visual Studio is an example of a suitable product available commercially. This information is given for the convenience of users of this document and does not constitute an endorsement by IEC of this product.

NOTE Certain operating system managed folders require specific access rights, for example, modifications in folder "Program Files" require "Administrator" rights. The Windows operating system provides several means to allow an application running with restricted user rights to execute actions with administrator privileges transparent to the user, for example, special restriction handling for identified directories, services with administration rights, executables that are configured to automatically run with administration rights. The alternative is to copy UIP executables into folders writeable for "normal" users.

4.7 UIP Lifecycle

4.7.1 General

The UIP state machine, outlined in IEC 62769-4, is composed of the Loaded, Created, Operational, Deactivated and Disposed states. The mechanisms affecting state changes are described in 4.7.

After the FDI Client has stored the UIP executable on the FDI Client, the FDI Client loads the UIP Assemblies dynamically into the memory and executes the related logic by calling the corresponding FDI-specified interface functions.

Subclause 4.7 describes rules about how the FDI Client shall activate and deactivate the UIP.

4.7.2 UIP Assembly activation steps

4.7.2.1 Load

The FDI Client shall load the UIP executables by using the LoadFrom mechanism. The .NET framework provides System.Reflection.Assembly.LoadFrom for this purpose:

The LoadFrom mechanism behaves as follows.

- LoadFrom loads the Assembly addressed with the file path and also the referenced Assemblies located within same directory. The argument string assemblyFile shall contain the file name of the UIP executable. The file name of the UIP executable represents the StartElementName described in IEC 62769-4.
- If an Assembly is loaded with LoadFrom, and later an Assembly in the "load context" attempts to load the same Assembly by display name, then this load attempt fails.
- If an Assembly with the same identity is already loaded (for example, by another UIP), then LoadFrom returns the Assembly that has been loaded before, even if a different file path was specified. Even a different file name does not matter. Only the identity of the Assembly is relevant.
- If an Assembly is loaded with LoadFrom, and the probing path includes an Assembly with the same identity (for example, in the Global Assembly Cache or an application directory), then this Assembly is loaded, even if a different file path was specified.
- `LoadFrom` requires the permissions `FileIOPermissionAccess.Read` and `FileIOPermissionAccess.PathDiscovery`, or `WebPermission`, on the specified path.
- `LoadFrom` loads the assembly into the default Application Domain.
- If a native Assembly image (generated by ngen.exe) exists for the specified file path, then it is not used. The Assembly cannot be loaded as domain neutral, i.e. the Assembly cannot be shared between Application Domains.

This behavior enforces deployment rules as follows.

- Rules regarding Assembly dependencies (see 4.7.2.4.2).

The FDI Client shall only use `LoadFrom`. The use of other .NET Assembly loading/object creation means is not allowed.

- Rules regarding shared Assemblies (see 4.7.2.4.3).
- A pre-compiled processor-specific machine code cannot be used.

- The security aspects regarding loading and execution of Assemblies are described in 4.9.

4.7.2.2 Create

Creating an instance of the UIP Assembly works using the .net library functions `System.Reflection.Assembly.GetTypes` and `System.Activator.CreateInstance`. The FDI type library declares a "custom attribute" named `UIPActivationClass`. This attribute shall only be added to the object implementing the interface `IDtmUiFunction` that actually implements the UIP start-up function. The attribute `UIPActivationClass` shall be used once only.

The FDI Client can now use `System.Reflection` services to clearly determine the UIP implemented activation procedure.

NOTE 1 Function System.Reflection.Assembly.GetTypes can be used to query the interface IDtmUiFunction.

NOTE 2 Function System.Attribute.GetCustomAttributes can be used for reading the additional custom attributes.

NOTE 3 The result of function invocation System.Activator.CreateInstance is an object of type IDtmUiFunction.

A data-type cast is needed.

4.7.2.3 Activate

Invocation of function `IDtmUiFunction.Init` finally activates the UIP for the user.

4.7.2.4 External libraries

4.7.2.4.1 General

UIP Assemblies can depend on external libraries (third-party libraries) and other Assemblies, for example, specific user control libraries. FDI Clients do not perform installation of UIPs, rather they dynamically load and execute the UIP. To support this usage, as well as the requirement to prevent possible problems of conflicting Assemblies, rules are specified for external libraries.

External libraries shall:

- be contained within the FDI Package;
- not require Microsoft Installer (MSI) installation;
- not require entries in the Windows Registry or the Global Assembly Cache;
- adhere to the access restrictions described in 4.9.2;
- be compatible with the platforms described in 4.1.2.

4.7.2.4.2 Loading of external libraries

The FDI Client loads the UIP Assembly, containing the UIP main class implementing interface `IDtmUiFunction`, by invocation of the .NET framework function `LoadFrom`. Referenced Assemblies that are stored in the same directory are automatically loaded together with this .NET Assembly. Referenced Assemblies that are stored in other locations (for example, in a sub-directory) have to be loaded explicitly by the UIP itself.

The UIP shall load such Assemblies also by invocation of the .NET framework function `LoadFrom`. Loading Assemblies with other .NET framework methods is not allowed.

Usage of external libraries shall not break the self-containment requirement for FDI Packages; all external libraries shall be included in the FDI UIP Package.

4.7.2.4.3 Loading of shared external libraries

An external library is a shared external library if a related .NET Assembly identity can be used from different UIP executables. The identity of a .NET Assembly matters. Installation path and Assembly filename are not relevant.

Usage of shared libraries shall not break the self-containment requirement for FDI Packages. Each of the delivered FDI Packages shall be shipped with all required UIP related libraries. The sharing mechanism comes from the .NET framework implemented optimization mechanism.

If a shared Assembly is used, then the following rules apply.

- Any incompatible change to the shared Assembly shall lead to a new identity, for example, different version number.
- Shared Assemblies shall not presume to be loaded from a specific installation path, for example, rely on the fact that some files are stored in the same directory or in a sub-directory.
- Static variables in shared Assemblies are also shared if the Assembly is loaded into the same Application Domain. Thus, static variables shall not have side effects in such scenarios. External shared libraries shall not declare static variables.
- Because of the self-containment rule defined for the FDI Package, shared Assemblies shall be deployed with all FDI Packages using a shared Assembly.

4.7.2.5 UIP Constructor invocation

Constructor and destructor implementation shall not throw exceptions. The constructor logic shall be limited to instantiate the object in terms of the internal data structure. The destructor logic shall be limited to destroy the object in terms of releasing memory resources. The constructor and the destructor shall not:

- invoke any call-back to the FDI Client,
- invoke any user interaction.

4.7.3 UIP Assembly deactivation steps

4.7.3.1 Deactivate

For UIP deactivation the FDI Client shall call the interface `IDtmUiFunction.BeginClose` and `IDtmUiFunction.EndClose`. On successful execution the UIP shall release all resources and the FDI Client shall delete all references to the UIP instance. The .NET garbage collector finally disposes the UIP runtime object.

4.7.3.2 Dispose

A .NET Assembly that is loaded into a process respectively into the related `ApplicationDomain` is never unloaded, except if the `ApplicationDomain` itself is destroyed. That means if the FDI Client loads a UIP Assembly into the default `ApplicationDomain`, then these Assemblies and all dependent Assemblies are never unloaded unless the application is closed.

The UIP Assemblies shall be developed with this .NET framework behavior in mind. To reduce the memory consumption, the following rules apply.

- Minimize the use of static variables, because these increase the memory consumption of the Assembly.
- Move UIP functionality that is not always (or rarely) needed to separate Assemblies. These Assemblies are then only automatically or manually loaded when the corresponding code is executed.

- Use shared Assemblies whenever possible.
- The FDI Client can execute .NET Assemblies in a separate Application Domain in order to have the ability to unload them.

4.8 Interaction between an FDI Client and a UIP

4.8.1 Handling of standard UI elements

UIPs shall delegate the presentation and handling of standard UI elements to the FDI Client. The standard UI elements are:

- UI Actions with standardized semantics (Apply/Close/Online Help), and
- UIP Specific status information.

To ensure a consistent user interface interaction across UIPs from different vendors, a UIP may delegate presentation and handling of additional UIP specific actions to the FDI Client. Nonetheless, UIPs are allowed to implement non-standard UI actions within their own UI area.

The set of standard UI actions and their respective semantics is fixed. However, the availability of these actions may change at any time depending on the internal state of the UIP. The set of additional UIP specific actions and their individual availability is not fixed. A UIP may add, remove, rename, enable or disable the UIP specific actions at any time depending on its requirements. The UIP has to inform the FDI Client whenever the availability of its standard actions or UIP specific actions changes (see events `IStandardActions.StandardActionItemSetChanged` and `IApplicationSpecificActions. ApplicationSpecificActionItemSetChanged)`.

An FDI Client may use dedicated UI elements, e.g. button controls, to provide direct access to the standard actions, as well as indirectly invoke them in the context of user interaction with other FDI Client UI elements. FDI Client shall always show all custom actions exposed by a UIP with dedicated UI elements.

4.8.2 Non-blocking service execution

4.8.2.1 FDI Client internal functions

The implementation of function Begin*OperationName* shall copy the content of Argument `asyncState` into member `AsyncState` of the returned `IAsyncResult` object.

The productive (time-consuming) part of the function named *OperationName* shall be performed in a different thread. The synchronization with the calling thread is handled via the `AsyncWaitHandle` object (class `WaitHandle`), which is also a member of the `IAsyncResult` object.

When processing of the productive part of the function named *OperationName* has finished, the `IAsyncResult` objects attribute `IsCompleted` shall be set to `True`. If the `AsyncCallBack` argument value is valid (not equal NULL), the FDI Client notifies the UIP using the callback.

The implementation of Cancel*OperationName* uses the argument `IAsyncResult` to identify the service that has been started with Begin*OperationName*. If Begin*OperationName* started an OPCUA service, the FDI Client shall call the OPCUA defined Cancel service.

4.8.2.2 UIP internal functions

The management of multiple asynchronous services in parallel shall be managed using the `AsyncState` object.

The `IAsyncResult` object returned by Begin*OperationName* contains the `WaitHandle` object. The UIP shall perform its own thread synchronization using the `WaitHandle` object.

4.8.2.3 Non-blocking service execution sequence

The following shows the interaction sequence between the FDI Client and the UIP. The thread management mechanisms implemented inside the FDI Client are not shown. The Interaction between an FDI Client and an FDI Server is based on Request/Response pattern. The FDI Client service request matches with the Begin*OperationName*. The AsyncCallback invocation matches with receiving the Client service response. End*OperationName* conveys the response contained results. Implementation of the non-blocking service execution does not require any thread management inside the FDI Client. Figure 4 shows an example of an IAsyncPattern based Asynchronous service execution.

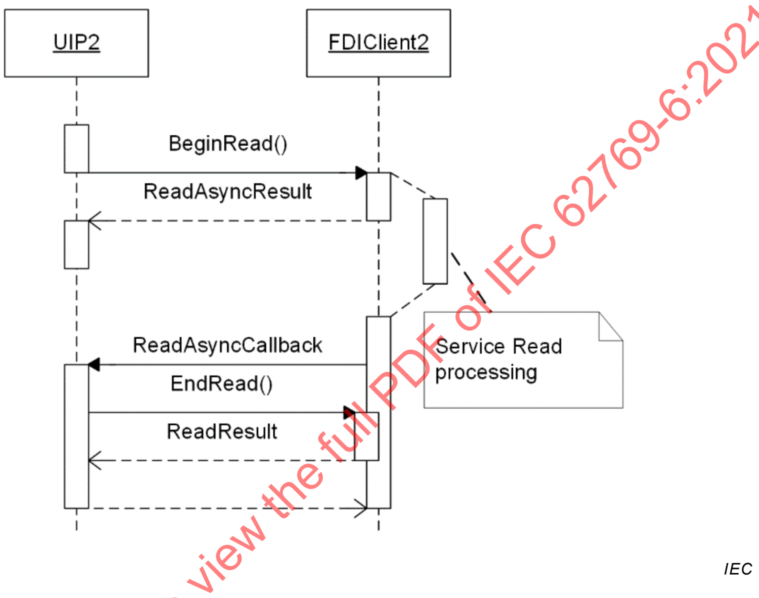
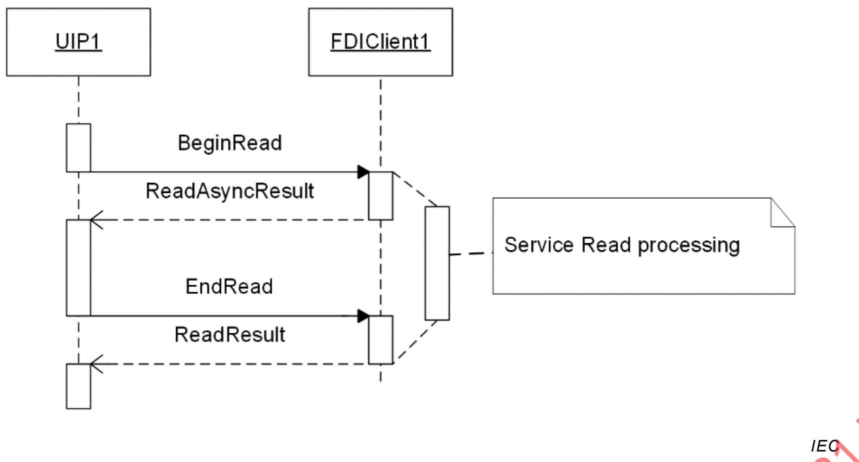


Figure 4 – IAsyncPattern based asynchronous service execution example

4.8.3 Blocking service execution

The FDI Client-provided interfaces allow performing synchronous Information Model access by using the functionality described in 4.8.1 in a way shown in Figure 5.



ReadAsyncResult is the object implementing the interface IAsyncResult.

Figure 5 – Blocking service execution example using IAsyncResult based pattern

4.8.4 Cancel service execution

Some services specified for the interface IDeviceModel (see Table 3) support canceling a started service by means of the function Cancel*OperationName.* The following Figure 6 will illustrate the processing sequence based on the Read service example.

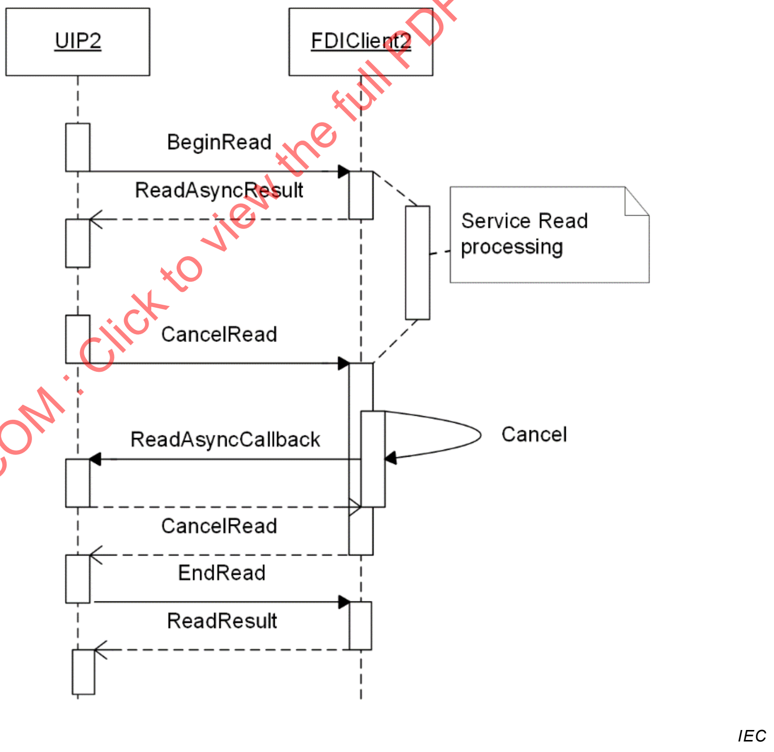


Figure 6 – Cancel service processing sequence example

The invocation of CancelRead triggers the FDI Client internal functions needed to cancel the active read operation. The FDI Client may not be able to cancel the operation immediately, but it should do so as soon as possible. Once the operation has been cancelled, the FDI Client notifies the UIP through the ReadAsyncCallback. The UIP shall then call the EndRead function.

NOTE A general challenge implementing this pattern is to handle race conditions properly on both sides (UIP2 and FDIClient2). If the FDI Client has forwarded the service execution via an OPC UA service, the actual service execution will run inside the FDI Server.

Depending on how the UIP is hosted, there may be three independently working processes. Therefore, the cancel request (sent by the UIP) may appear right after the FDI Server has already finished the service request. The related response sent by the FDI Server may have arrived at the FDI Client (or not). The FDI Client may invoke the ReadAsyncCallBack while the UIP invokes the CancelRead.

`ReadAsyncResult` is the object implementing the interface `IAsyncResult`.

4.8.5 Threading

4.8.5.1 Implementation rules

The UIP shall be able to receive calls in any thread.

The UIP shall not block the calls coming from the FDI Client.

The UIP shall not use the FDI Client thread to signal back the callback to the FDI Client itself. This is to prevent deadlocks and endless loops.

The UIP shall not run synchronous operations as described in 4.8.3 in the user interface thread: the user interface thread of a process shall be dedicated to receiving user inputs and perform drawing tasks only.

The UIP and FDI Client shall not block the user interface thread. The user interface shall always stay responsive. The user interface thread is shared between the different FDI user interface related objects for user input and drawing operations. If one object blocks this thread in order to perform some processing, this would affect the responsiveness of other user interfaces.

The UIP and FDI Client shall not block a Begin*OperationName* method call: a Begin*OperationName* method shall only start an asynchronous operation. The caller shall not be blocked.

4.8.6 Timeout

The interfaces referred in Clause 5 enable asynchronous service execution. The time for the execution of such services depends on performance constraints related to: bus communication, FDI Client/FDI Server performance. The rules listed below target the system interoperability regarding the prevention of "Race Conditions". The general rule is that the component is allowed to manage timeout handling only for those processes that are completely under the control of that component. The following list shows which elements of the entire system are allowed to implement the timeout detection function.

- UIP: The UIP shall not implement timeout detection.
- Business Logic: The Business Logic shall not implement timeout detection (FDI Package).
- FDI Client: The FDI Client shall implement timeout detection. In the case of OPC UA, the related support is built into the OPC UA communication stacks. Timeout detected during operations performed on behalf of the UIP shall be forwarded as negative function result codes.
- FDI Server: The FDI Server shall implement timeout detection. In the case of OPC UA, the related support is built into the OPC UA communication stacks.

- Communication Server: The Communication Server implements timeout detection for the OPC UA connection according to the OPC UA Specification. Related support is built into the OPC UA communication stacks. Additionally, the Communication Server implements timeout detection limited to the network directly connected to the physical port connected to the Communication Server.

4.8.7 Exception handling

An important specification goal is to make a clear distinction between software quality problems and anticipated processing states. Therefore, the specification defines two general exception categories:

a) Exceptions that indicate software states or events that have not been anticipated during the software development are considered as software quality issues (Run-time error).

b) Exceptions indicating anticipated software operation failures.

Examples of software quality issues indicated by exceptions are:

- function argument type mismatch;
- function argument value range mismatch;
- division by zero;
- NULL Pointer reference.

Examples of anticipated error handling are:

- communication problem handling;
- general IO data processing;
- user input errors.

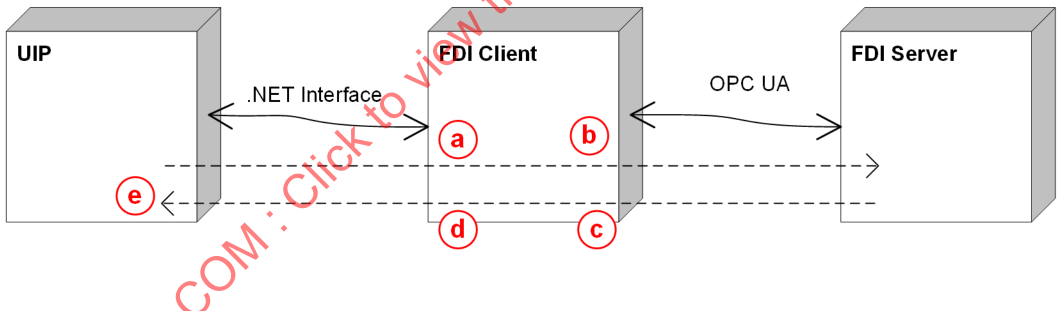


Figure 7 – Exception source

According to the FDI Architecture, exceptions can occur in different steps of the service processing, see Figure 7:

a) passing the request from the UIP to the FDI Client:

b) request forwarding inside the FDI Client;

c) processing the response from the FDI Server;

d) forwarding the response to the UIP;

e) response processing inside the UIP.

Service processing problems detected inside the FDI Server and beyond are handled through OPC UA defined service results.

Regarding the implementation of the `IAsyncResult` pattern, the following rules apply.

– Any failure occurring with step a) shall be reported by an exception thrown by the Begin*OperationName*.

– Any failure occurring during steps b) to e) shall be handled by the corresponding component. The execution of the End*OperationName* shall then report the failure via an exception.

4.8.8 Type safe interfaces

The Information Model hosts device variables of different types. The values of such variables are transferred using the class DataValue (FDI Interfaces and Data Types.CHM).

The Device Access Services support writing or reading multiple variables within one service. The data type chosen for data transport is `DataValue` implementing the type safe transport because of the `DataValue` property `Datatype` describing the value data type by means of `Datatype` enumeration. Because the `DataValue` property Value get/set functions use data type Object to convey the actual value, the data receiver (UIP) shall verify the data type before data processing.

4.8.9 Globalization and localization

UIP localization support can be implemented through resource files (.res(x)) or satellite Assemblies.

The FDI Client shall set the locale and country information that shall be used by the UIP by means of the arguments `currentRegion` and `currentCulture` that are submitted with the invocation of method `Fdi.Dtm.Ui.IDtmUiFunction.Init`. The UIP shall not derive locale information via the `Thread.CurrentUICulture`. The data type for `currentRegion` is `RegionInfo` defined in the .NET namespace `System.Globalization`. The data type for `currentCulture` is `CultureInfo` defined in the .NET namespace `System.Globalization`.

4.8.10 WPF Control handling

If a UIP implementation is based on WPF `UserControl`, the UIP inherits the interface from the class `UserControl`, which means there will be more methods attributes and events available for the FDI Client that are not covered by the FDI specification. Conversely, a UIP implements the accessibility function. The related rules affect, on the one hand, the quality of the UIP product and, on the other hand, the interoperability.

4.8.11 Win Form handling

If a UIP implementation is based on Windows Forms, the UIP inherits from the class `UserControl`, which means there will be more methods attributes and events available for the FDI Client that are not covered by the FDI specification. Conversely, a UIP implements the accessibility function. The related rules affect, on the one hand, the quality of the UIP product and, on the other hand, the interoperability. The FDI Client shall make thread-safe calls to the `Windows.Forms` controls.

4.9 Security

4.9.1 General

The goal of security is to protect a system against threats compromising the system's stability, integrity and sensitive data.

System-wide security begins with the design process, which is out of the scope of standardization. From the system's perspective, security is about controlling access to resources, such as application components, data, and hardware. The .NET framework provides support for constraining access to resources. The system security is based on control over access permissions.

A different approach is based on certification and authentication. Since all FDI Packages need compliance testing and certification, the presumption is that such certified FDI Packages don't pose any threats to a system. This means that a UIP could be executed with fully trusted permissions.

While an over-constrained system could lead into functional problems, unconstrained permissions can be seen as security threats. Subclause 4.9 represents a compromise between both ways.

4.9.2 Access permissions

4.9.2.1 General

Within 4.9.2, technology-specific permissions and restrictions are specified in addition to the one specified in IEC 62769-2. The permissions and restrictions shall be enforced by the FDI Client. The implementation rules define how this shall be implemented.

4.9.2.2 Technology specific UIP permissions and restrictions

The general UIP permissions and restrictions are specified in IEC 62769-2. The permissions and restrictions specified in 4.9.2 are specific to .NET-based UIPs.

a) Launching of an Active-X component is not allowed.

b) A UIP shall not write to the operating system registry. Read access to registry is allowed.

The FDI Client shall restrict the UIP permissions in accordance with this list.

4.9.2.3 Implementation rules

In order to restrict the permissions of a UIP, an FDI Client shall execute a UIP in a separate process (not the FDI Client process) that should run with a separate user account (4.5.2 describes how an FDI Client can manage hosting a UIP in a different process). By restricting the permissions of the separate user account, the FDI Client also restricts the permissions of the UIP running under the separate user account.

How the separate user account is created and configured is host specific. That includes

– whether it is a local or a domain account;
– whether it is used only for UIP hosting or also for other purposes;
– whether it is reused for all UIP processes of the same FDI Client instance or several FDI Clients;
– how restrictions are implemented (e.g. limited access to the file system by mechanism of the operating system, blocking network access by a firewall).

Whether the FDI Client executes each UIP instance in its own process, or executes several or all UIP instances in the same process, is FDI Client-specific.

As UIPs run in a separate process, UIP developers need to be aware that opening a modal dialog with operating system mechanisms will only provide modality within the Windows of that process. To avoid this behavior, UIP developers can start the modal UIP or start a new modal UIP.

4.9.3 Code identity concept

The ability to uniquely identify UIP executables contributes to the system's security.

As earlier described in 4.3, UIP executables shall be signed with strong names. Strong-named signed .NET Assemblies enable:

– unique identification of UIP executables;

– code integrity verification.

NOTE The benefit of strong-named UIP executables is lost if this Assembly dynamically loads other library Assemblies that are not signed with strong names.

5 Interface definition

Table 2 to Table 9 specify the mapping between the abstract services specified in IEC 62769-2 and the corresponding .NET implementation, which is found in the type library file "FDI.DLL" and a related help file "FDI Interfaces and Data Types.chm".

NOTE The Files "FDI.DLL" and "FDI Interfaces and Data Types.chm" can be obtained from the fieldbus organizations. (See also http://www.fdi-cooperation.com).

Table 2 specifies the mapping of the Base Property Services.

Table 2 – Base Property Services

Abstract Service	.NET Implementation
GetDeviceAccessInterfaceVersion	IDeviceAccess.Version
GetOnlineAccessAvailability	IDeviceAccess.OnlineAccessAvailable

Table 3 specifies the mapping of the Device Model Services.

Table 3 – Device Model Services

Abstract Service	.NET Implementation
Browse	IDeviceModel.BeginBrowse IDeviceModel.CancelBrowse IDeviceModel.EndBrowse
Read	IDeviceModel.BeginRead IDeviceModel.CancelRead IDeviceModel.EndRead
Write	IDeviceModel.BeginWrite IDeviceModel.CancelWrite IDeviceModel.EndWrite
CreateSubscription	IDeviceModel.BeginCreateSubscription IDeviceModel.EndCreateSubscription
Subscribe	IDeviceModel.BeginSubscribe IDeviceModel.EndSubscribe
Unsubscribe	IDeviceModel.BeginUnsubscribe IDeviceModel.EndUnsubscribe
DeleteSubscription	IDeviceModel.BeginDeleteSubscription IDeviceModel.EndDeleteSubscription
DataChangeCallback	DataChangeCallback

Table 4 specifies the mapping of the Access Control Services.

Table 4 – Access Control Services

Abstract Service	.NET Implementation
InitLock	IAccessControl.BeginInitLock IAccessControl.EndInitLock
ExitLock	IAccessControl.BeginExitLock IAccessControl.EndExitLock

Table 5 specifies the mapping of the Direct Access Services.

Table 5 – Direct Access Services

Abstract Service	.NET Implementation
InitDirectAccess	IDirectAccess.BeginInitDirectAccess IDirectAccess.EndInitDirectAccess
ExitDirectAccess	IDirectAccess.BeginExitDirectAccess IDirectAccess.EndExitDirectAccess
Transfer	IDirectAccess.BeginTransfer IDirectAccess.EndTransfer

Table 6 specifies the mapping of the Hosting Services.

Table 6 – Hosting Services

Abstract Service	.NET Implementation
GetClientTechnologyVersion	Fdi.Frame.IFrame.Version
OpenUserInterface[a)]	Fdi.Frame.Ui.IFrameUi.BeginOpenDtmUiModal Fdi.Frame.Ui.IFrameUi.EndOpenDtmUiModal Fdi.Frame.Ui.IFrameUi.BeginOpenDtmUi Fdi.Frame.Ui.IFrameUi.EndOpenDtmUi
CloseUserInterface[a)]	Fdi.Dtm.Ui.CloseMeRequestHandler [b)] Fdi.Frame.Ui.IFrameUi.BeginCloseDtmUi Fdi.Frame.Ui.IFrameUi.EndCloseDtmUi
LogAuditTrailMessage	Fdi.Frame.IAuditTrail.Notify
SaveUserSettings	Fdi.Frame.IUserSettings.SaveUserSettings
LoadUserSettings	Fdi.Frame.IUserSettings.LoadUserSettings
Trace	Fdi.Frame.ITrace.TraceData Fdi.Frame.ITrace.TraceEvent
ShowMessageBox	Fdi.Frame.Ui.IFrameUi.ShowMessageBox
ShowProgressBar	Fdi.Frame.Ui.IFrameUi.ShowProgress
CancelCallback	Fdi.Frame.Ui.CancelEventHandler
UpdateShowProgressBar	Fdi.Frame.Ui.IProgressUi.UpdateProgress
EndShowProgressBar	Fdi.Frame.Ui.IProgressUi.EndProgress
DefaultResult	System.Windows.MessageBoxResult
ButtonSet	System.Windows.MessageBoxButton
AcknStyle	System.Windows.MessageBoxImage

Abstract Service	.NET Implementation
ExportFile	Fdi.Frame.ImportExport.IFileExport.BeginInitExport Fdi.Frame.ImportExport.IFileExport.EndInitExport Fdi.Frame.ImportExport.IFileExport.BeginWriteExport Fdi.Frame.ImportExport.IFileExport.EndWriteExport Fdi.Frame.ImportExport.IFileExport.BeginFinishExport Fdi.Frame.ImportExport.IFileExport.EndFinishExport
CancelExportFile	Fdi.Frame.ImportExport.IFileExport.BeginFinishExport Fdi.Frame.ImportExport.IFileExport.EndFinishExport
ImportFile	Fdi.Frame.ImportExport.IFileExport.BeginInitImport Fdi.Frame.ImportExport.IFileExport.EndInitImport Fdi.Frame.ImportExport.IFileExport.BeginReadImport Fdi.Frame.ImportExport.IFileExport.EndReadImport Fdi.Frame.ImportExport.IFileExport.BeginFinishImport Fdi.Frame.ImportExport.IFileExport.EndFinishImport
CancelImportFile	Fdi.Frame.ImportExport.IFileExport.BeginFinishImport Fdi.Frame.ImportExport.IFileExport.EndFinishImport
OpenDefaultApplication	Fdi.Frame.DefaultApplication.IOpenDefaultApplication.BeginInitOpenDefaultApplication Fdi.Frame.DefaultApplication.IOpenDefaultApplication.EndInitOpenDefaultApplication Fdi.Frame.DefaultApplication.IOpenDefaultApplication.BeginWriteOpenDefaultApplication Fdi.Frame.DefaultApplication.IOpenDefaultApplication.EndWriteOpenDefaultApplication Fdi.Frame.DefaultApplication.IOpenDefaultApplication.BeginFinishOpenDefaultApplication Fdi.Frame.DefaultApplication.IOpenDefaultApplication.EndFinishOpenDefaultApplication
GetEnvironmentProperties	Fdi.Frame.HostProperty. GetHostingProperties

[a] Functions `OpenUserInterface`, `CloseUserInterface` shall only be started using the operation pattern described in 4.8.2.3.

[b] To be used by the UIP to close itself.

Table 7 specifies the mapping of the UIP Services.

Table 7 – UIP Services

Abstract Service	.NET Implementation
Activate	Fdi.Dtm.Ui.IDtmUiFunction.Init
Deactivate	Fdi.Dtm.Ui.IDtmUiFunction.BeginClose [a)] Fdi.Dtm.Ui.IDtmUiFunction.EndClose [a)]
SetSystemLabel	Fdi.Dtm.Ui.IDtmUiFunction.SystemGuiLabel
SetTraceLevel	Fdi.Dtm.Ui.IDtmUiFunction.TraceLevel
TraceLevel	Fdi.Frame.TraceEventType
InvokeStandardAction	Fdi.Dtm.Ui.IStandardActions.InvokeStandardAction
InvokeApplicationSpecificAction	Fdi.Dtm.Ui. IApplicationSpecificActions.InvokeApplicationSpecificAction
GetStandardActionItems	Fdi.Dtm.Ui.IStandardActions.ActionItemSet
GetApplicationSpecificActionItems	Fdi.Dtm.Ui. IApplicationSpecificActions.ApplicationSpecificActionItemSet
StandardActionItemsChangeCallback	Fdi.Dtm.Ui.IStandardActions.StandardActionItemSetChanged
ApplicationSpecificActionItemsChangeCallba ck	Fdi.Dtm.Ui. IApplicationSpecificActions.ApplicationSpecificActionItemSetChan ged

[a)] The Deactivate service specified response `deactivateCancelled` maps to the exception `FdiCannotCloseUiException` to be thrown by the UIP if the UIP has problems with the deactivation.

Table 8 specifies the mapping of the base data types.

Table 8 – Base Data Types

Base data type	.NET Implementation	
Boolean	Fdi.DataTypes.BooleanValue	enum DataType.Boolean
String	Fdi.DataTypes.StringValue	enum DataType.String
ByteString	Fdi.DataTypes.BinaryValue	enum DataType.Binary
UtcTime	Fdi.DataTypes.DateTimeValue	enum DataType.DateTime
Int8	Fdi.DataTypes.SByteValue	enum DataType.SByte
Int16	Fdi.DataTypes.ShortValue	enum DataType.Short
Int32	Fdi.DataTypes.IntValue	enum DataType.Int
Int64	Fdi.DataTypes.LongValue	enum DataType.Long
Byte	Fdi.DataTypes.ByteValue	enum DataType.Byte
UInt16	Fdi.DataTypes.UShortValue	enum DataType.UShort
UInt32	Fdi.DataTypes.UIntValue	enum DataType.UInt
UInt64	Fdi.DataTypes.ULongValue	enum DataType.ULong
Float	Fdi.DataTypes.FloatValue	enum DataType.Float
Double	Fdi.DataTypes.DoubleValue	enum DataType.Double
Duration	Fdi.DataTypes.TimeSpanValue	enum DataType.TimeSpan

Table 9 specifies the mapping of the special data types.

Table 9 – Special Types

Special Data type	**.NET Implementation**	
Attribute Ids	Fdi.DeviceAccess.AttributeType	
Variant	Fdi.DeviceAccess.DataValue	
NodeSpecifier	Fdi.DeviceAccess.NodeSpecifier	
Data Value	Fdi.DeviceAccess.ReadResult	
Localized Text	Fdi. DataTypes.LocalizedTextValue	enum DataType.LocalizedText
Range	Fdi. DataTypes.RangeValue	enum DataType.Range
EU Information	Fdi. DataTypes.EUInfoValue	enum DataType.EngineeringUnit
Enum Value	Fdi. DataTypes.EnumValue	enum DataType.Enumerator
InnerErrorInfo	Fdi.DeviceAccess.InnerErrorInfo	
NumericRange	Fdi.DeviceAccess.ArrayIndexRange	

Data arrays can be conveyed using class `Fdi.DataTypes.ArrayValue`.

Detailed interface definition and interface documentation are available in:

– FDI.DLL (.NET Assembly)
– FDI Interfaces and Data Types.CHM (Help File)

SOMMAIRE

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

____

INTÉGRATION DES APPAREILS DE TERRAIN (FDI) –

Partie 6: Mapping de technologies

AVANT-PROPOS

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

La Norme internationale IEC 62769-6 a été établie par le sous-comité 65E: Les dispositifs et leur intégration dans les systèmes de l'entreprise, du comité d'études 65 de l'IEC: Mesure, commande et automation dans les processus industriels.

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

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

a) redéfinition du concept de sécurité pour l'exécution de l'UIP.

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

FDIS	Rapport de vote
65E/763/FDIS	65E/773/RVD

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

Ce document a été rédigé selon les Directives ISO/IEC, Partie 2.

Une liste de toutes les parties de la série IEC 62769, publiées sous le titre général *Intégration des appareils de terrain (FDI)*, peut être consultée sur le site web de l'IEC.

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

- reconduit,
- supprimé,
- remplacé par une édition révisée, ou
- amendé.

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

INTRODUCTION

La série IEC 62769 est publiée sous le titre général "*Intégration des appareils de terrain (FDI)*" et comporte les parties suivantes:

- Partie 1: Vue d'ensemble
- Partie 2: Client FDI
- Partie 3: Serveur FDI
- Partie 4: Paquetages FDI
- Partie 5: Modèle d'Information FDI
- Partie 6: Mapping de technologies FDI
- Partie 7: Appareils de Communication FDI
- Partie 100: Profils – Extensions de protocoles génériques
- Partie 101-1: Profils – Foundation Fieldbus H1
- Partie 101-2: Profils – Foundation Fieldbus HSE
- Partie 103-1: Profils – PROFIBUS
- Partie 103-4: Profils – PROFINET
- Partie 109-1: Profils – HART et WirelessHART
- Partie 115-2: Profils – Définitions spécifiques au protocole pour Modbus-RTU
- Partie 150-1: Profils – ISA 100.11a

INTÉGRATION DES APPAREILS DE TERRAIN (FDI) –

Partie 6: Mapping de technologies

1 Domaine d'application

La présente partie de l'IEC 62769 spécifie le mapping de technologies pour les concepts décrits dans la norme d'intégration des appareils de terrain (FDI). Le mapping de technologies porte essentiellement sur la mise en œuvre relative aux composants: Client FDI et Plugiciel d'Interface Utilisateur (UIP) qui ne sont spécifiques qu'à la plate-forme WORKSTATION (Poste de travail)/.NET telle que définie dans l'IEC 62769-4.

2 Références normatives

Les documents ci-après, dans leur intégralité ou non, sont des références normatives indispensables à l'application du présent document. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

IEC 61804 (toutes les parties), *Blocs fonctionnels (FB) pour les procédés industriels et le Langage de Description Electronique de Produit (EDDL)*

IEC 62769-1, *Intégration des appareils de terrain (FDI) – Partie 1: Vue d'ensemble*

IEC 62769-2, *Intégration des appareils de terrain (FDI) – Partie 2: Client FDI*

IEC 62769-4, *Intégration des appareils de terrain (FDI) – Partie 4: Paquetages FDI*

IEC 62541 (toutes les parties), *Architecture unifiée OPC*

ISO/IEC 19505-1, *Information technology – Object Management Group Unified Modeling Language (OMG UML) – Part 1: Infrastructure* (disponible en anglais seulement)

ISO/IEC 29500 (toutes les parties), *Information technology – Document description and processing languages – Office Open XML File Formats* (disponible en anglais seulement)

3 Termes, définitions, termes abrégés, symboles et conventions

3.1 Termes et définitions

Pour les besoins du présent document, les termes et définitions de l'IEC 62769-1, ainsi que les suivants s'appliquent.

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

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

3.1.1
Domaine d'Application
environnement isolé dans lequel s'exécutent les applications

3.1.2
Bibliothèque de Types FDI
assemblage qui contient les interfaces et les types de données qui sont utilisés pour l'échange de données et l'interaction entre un UIP et un Client FDI

3.1.3
Cache Global d'Assemblages
cache de codes à l'échelle de la machine qui stocke des Assemblages spécifiquement désignés pour être partagés par plusieurs applications

3.1.4
Registre Windows
base de données définie par le système dans laquelle les applications et les composants système stockent et récupèrent les données de configuration

3.2 Termes abrégés

Pour les besoins du présent document, les termes abrégés de l'IEC 62769-1, ainsi que les suivants s'appliquent.

CLR	Common Language Run-time (moteur d'exécution du langage commun)
MSI	Microsoft Installer (programme d'installation Microsoft)
WPF	Windows Presentation Foundation (fondation de présentation Windows)
UML	Unified Modeling Language (langage de modélisation unifié)

3.3 Symboles

Les figures du présent document utilisent des symboles graphiques conformément à l'ISO/IEC 19505-1 (UML 2.0).

3.4 Conventions

Pour les besoins du présent document, les conventions de l'IEC 62769-1 s'appliquent.

La description de l'exécution d'un service sans blocage en 4.8.2 utilise des caractères italiques pour identifier le nom d'une opération générique à laquelle la fonction interne est appliquée.

4 Concepts techniques

4.1 Généralités

4.1.1 Vue d'ensemble

En 4.1.2, 4.2, 4.3, 4.4 et 4.5, le présent document décrit tout d'abord la base de la technologie pour la mise en œuvre de l'UIP, l'environnement matériel et logiciel, y compris les règles connexes de mise en œuvre. L'Article 4 suit une approche orientée cycle de vie (cas d'utilisation).

Le paragraphe 4.6 décrit les procédures de déploiement des copies et les règles connexes de mise en œuvre pour l'UIP et le Client FDI. L'instanciation et la terminaison de l'exécutable de l'UIP sont décrites en 4.7. Le paragraphe 4.8 définit les règles d'interaction entre un Client FDI et l'UIP. Les définitions relatives à la sécurité sont données en 4.9. Les définitions d'interface de service pour le Client FDI et l'UIP sont spécifiées à l'Article 5.

4.1.2 Plates-formes

L'UIP et le Client FDI doivent être construits sur le .NET Framework de Microsoft et exécutés dans le Common Language Run-time .NET.

L'ensemble minimal des appareils d'E/S pris en charge par le poste de travail est le suivant: une souris, un clavier et un écran couleur de résolution 1 024 × 768 pixels.

Le Tableau 1 suivant énumère toutes les technologies et leurs éditions conformes aux composants FDI.

Tableau 1 – Référence de l'édition de technologie

Technologie	Norme	Édition
.NET	N/A	CLR4 pour la mise en œuvre de l'UIP
EDDL	IEC 61804IEC 61804	2016
OPC UA (Parties 1-8)	IEC 62541	2015
Convention de Paquetage Ouvert (OPC)	ISO/IEC 29500	2016
Langage de balisage extensible (XML)	N/A	W3C, 1.0 (cinquième édition)

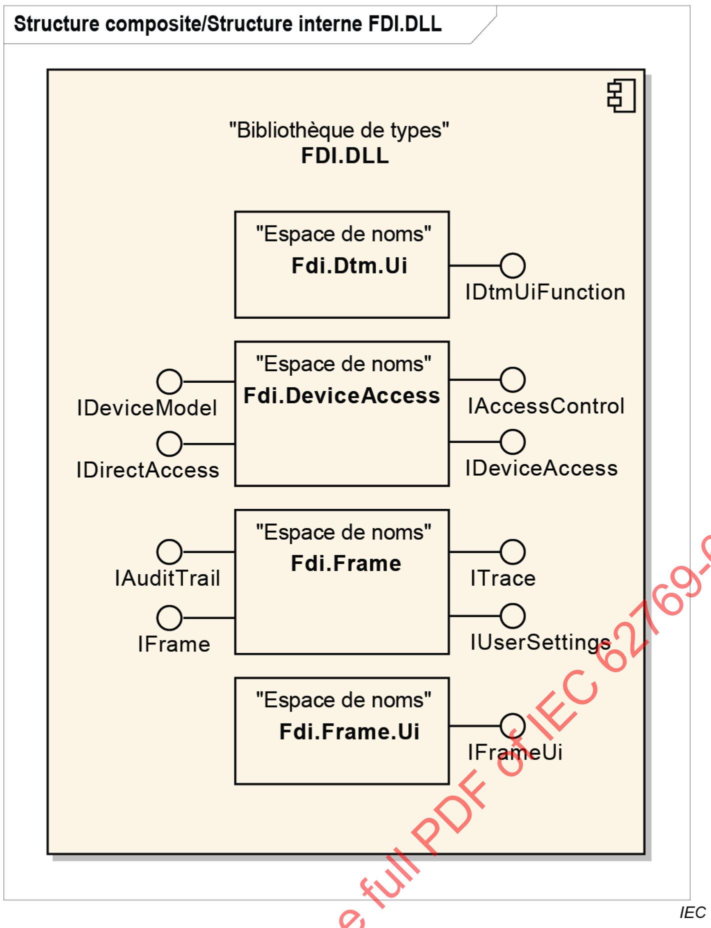
4.1.3 Bibliothèque de Types FDI

Les Services d'Accès à l'Appareil et les Services d'UIP peuvent être modélisés comme des interfaces .NET conformes aux arguments de types de données .NET. Ces interfaces et ces types de données sont utilisés pour l'échange de données et l'interaction entre l'UIP et le Client FDI. Pour l'objet relatif au traitement des erreurs d'exécution durant les appels de méthodes d'interfaces, les classes d'exceptions .NET sont définies.

Les interfaces, types de données et classes d'exception FDI .NET sont définis dans une Bibliothèque de Types FDI unique. La Bibliothèque de Types FDI est un Assemblage à nom fort. Le nom de fichier de cet Assemblage doit être ‘fdi.dll’. Le fichier fdi.dll doit être versionné conformément à l’IEC 62769-1:2020, 8.1. La Bibliothèque de Types FDI fait partie de la Technique de Base FDI conformément à l’IEC 62769-1:2020, 8.3.2.1. Par conséquent, elle influe directement sur la Version de Technologie FDI. Toutes les modifications compatibles du fichier fdi.dll se traduisent par une augmentation de la portion mineure de la Version de Technologie FDI. Les modifications incompatibles se traduisent par une augmentation de la portion majeure de la Version de Technologie FDI (voir l’IEC 62769-1:2020, 8.3.2.2).

La Bibliothèque de Types FDI est signée avec une clé unique par l’émetteur du fichier. La Bibliothèque de Types FDI doit être installée séparément dans le cadre de chaque installation de Client FDI. Des Plugiciels d’Interface Utilisateur (UIP) et l’Application du Client FDI doivent utiliser cette instance du fichier fdi.dll. Les UIP ne doivent pas transporter ou déployer la Bibliothèque de Types FDI. Il incombe au Client FDI de fournir des moyens d’autoriser les mises à jour de cette bibliothèque de types au fil du temps.

La Figure 1 représente la structure de Bibliothèque de Types FDI.



NOTE Le diagramme de structure composite ne représente que les interfaces de base qui mettent en œuvre les interfaces définies dans l'IEC 62769-2.

Figure 1 – Structure de la Bibliothèque de Types FDI

4.2 Représentation de l'UIP

La variante de l'UIP peut contenir un seul ou plusieurs modules exécutables (Assemblage .NET) et leurs fichiers supplémentaires connexes (par exemple: fichiers ressources). Le module exécutable de la Variante de l'UIP est appelé "exécutable de l'UIP". Le ou les fichiers supplémentaires de la Variante de l'UIP sont appelés "suppléments de l'UIP".

Le ou les suppléments de l'UIP sont stockés dans un ou plusieurs sous-dossiers du répertoire d'installation de l'exécutable de l'UIP.

EXEMPLE Fichiers de ressources et données de configuration d'application.

Le RuntimeId d'une Variante de l'UIP doit être ".NET Framework CLR4", voir l'IEC 62769-4. Les Clients FDI qui prennent en charge ce RuntimeId doivent prendre en charge le .NET Framework 4.6.1 ou supérieur utilisant le CLR4 et les UIP avec ce RuntimeId doivent utiliser le .NET Framework 4.6.1 ou inférieur qui prend en charge le CLR4 (c'est-à-dire: de .NET Framework 4.0 jusqu'à .NET Framework 4.6.1).

La Variante de l'UIP doit être autonome. Toutes les bibliothèques exigées de l'UIP (Assemblages .NET) exigées par une Variante de l'UIP sont stockées dans le même Dossier.

4.3 Représentation de l'exécutable de l'UIP

La mise en œuvre de l'UIP dépend du type d'éléments d'interface utilisateur qui peuvent être intégrés dans l'environnement d'hébergement de l'interface utilisateur du Client FDI. L'UIP doit être mis en œuvre comme suit: NET `System.Windows.Forms` classe `UserControl` ou Windows Presentation Foundation (WPF) `System.Windows.Controls` classe `UserControl`.