IEC 62769-6-100

Edition 1.0 2023-04

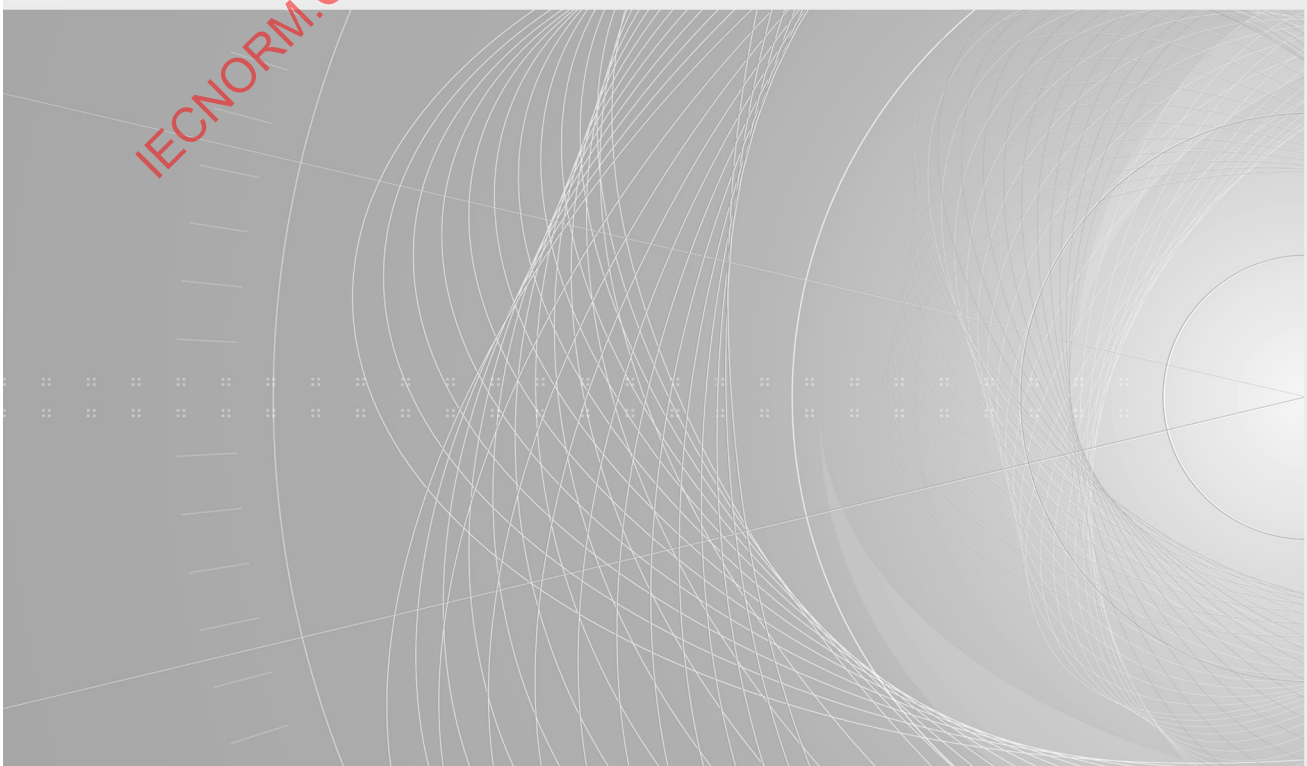
INTERNATIONAL STANDARD

NORME INTERNATIONALE

colour inside

Field Device Integration (FDI®) – Part 6-100: Technology Mapping – .Net

Intégration des appareils de terrain (FDI®) – Partie 6-100: Mapping de technologies – .Net



IEC 62769-6-100:2023-04(en-fr)

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

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

IEC 62769-6-100

Edition 1.0 2023-04

INTERNATIONAL STANDARD

NORME INTERNATIONALE

colour inside

Field Device Integration (FDI®) – Part 6-100: Technology Mapping – .Net

Intégration des appareils de terrain (FDI®) – Partie 6-100: Mapping de technologies – .Net

INTERNATIONAL ELECTROTECHNICAL COMMISSION

COMMISSION ELECTROTECHNIQUE INTERNATIONALE

ICS 25.040.40; 35.100.05

ISBN 978-2-8322-6809-4

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

_____

FIELD DEVICE INTEGRATION (FDI®) –

Part 6-100: Technology Mapping – .NET

FOREWORD

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

IEC 62769-6-100 has been prepared by subcommittee 65E: Devices and integration in enterprise systems, of IEC technical committee 65: Industrial-process measurement, control and automation. It is an International Standard.

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

Draft	Report on voting
65E/868/CDV	65E/925/RVC

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

The language used for the development of this International Standard is English.

This document was drafted in accordance with ISO/IEC Directives, Part 2, and developed in accordance with ISO/IEC Directives, Part 1 and ISO/IEC Directives, IEC Supplement, available at www.iec.ch/members_experts/refdocs. The main document types developed by IEC are described in greater detail at www.iec.ch/publications.

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 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 document 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.

FIELD DEVICE INTEGRATION (FDI®) –

Part 6-100: Technology Mapping – .NET

1 Scope

This part of IEC 62769 specifies the technology mapping for the concepts described in the Field Device Integration (FDI®[1]) standard. The technology mapping focuses on implementation regarding the components FDI® Client and User Interface Plug-in (UIP) using the Runtime .NET. This runtime is specific only to the WORKSTATION platform as defined in IEC 62769-4.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IEC 62769-1:2021, *Field device integration (FDI®) – Part 1: Overview*

IEC 62769-2, *Field device integration (FDI®) – Part 2: Client*

IEC 62769-4, *Field device integration (FDI®) – Part 4: FDI® Packages*

IEC 62769-6, *Field device integration (FDI®) – Part 6: Technology Mappings*

3 Terms, definitions, abbreviated terms, acronyms and symbols

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 terminology 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

[1] FDI is a registered trademark of the non-profit organization Fieldbus Foundation, Inc. This information is given for the convenience of users of this document 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.

3.1.3
FDI® Type Library
typescript file that contains the interfaces and data types that are used for the data exchange and interaction between a UIP and an FDI® Client

3.1.4
Global Assembly Cache
machine-wide code cache that stores Assemblies specifically designated to be shared by several applications

3.1.5
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, IEC 62769-6, as well as the following apply.

MSI Microsoft Installer

WPF Windows Presentation Foundation

UML Unified Modeling Language

3.3 Symbols

Figures in this document use the graphical symbols according to I'ISO/IEC 19505-1 (UML 2.0).

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 the technology base for UIP implementation based on the runtime .NET Framework CLR4, the hardware and software environment including the related implementation rules. Clause 4 follows a lifecycle (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 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 provided within a Nuget Package, which contains one or more strong named assemblies. The file name of this Nuget Package shall be Fdi.<version>.nupkg. The FDI® Type Library shall be versioned as per IEC 62769-1:2021, 8.1. The FDI® Type Library is part of the FDI® Core Technology as per IEC 62769-1:2021, 8.3.2.1. Therefore, it directly influences the FDI® Technology Version. All compatible changes of the FDI® Type Library 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 62769-1:2021, 8.3.2.2). The version information of the FDI® Type Library can be found in FCG TS10099.

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® Type Library. 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.

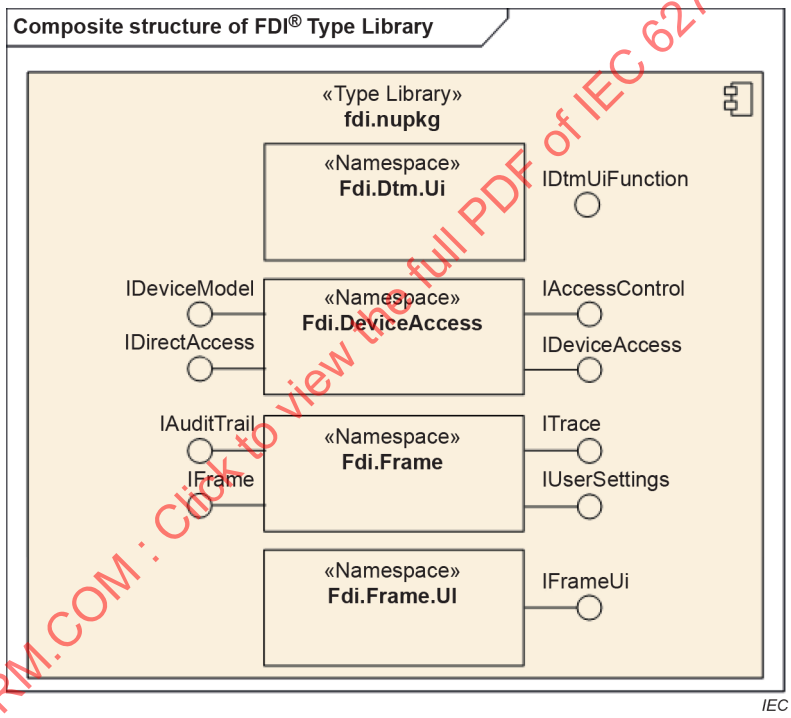


Figure 1 – FDI® Type Library structure

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

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 Examples of UIP supplementary data files include resource files and application configuration data.

The supported RuntimeIds and .NET Framework versions for a specific FDI® Technology Version are specified in FCG TS10099 FDI® Technology Management.

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 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 CLR versions

4.5.1 General

Specific CLR 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 able 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 in 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. In case 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.

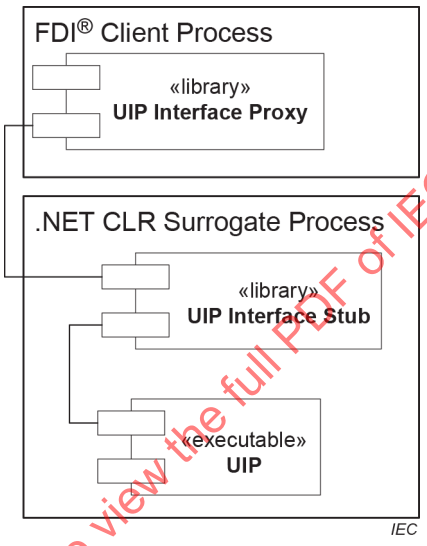


Figure 2 – .NET surrogate process

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®[2] 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.

2 Visual Studio is the trademark of Microsoft Corporation. This information is given for the convenience of users of this document and does not constitute an endorsement by IEC of the trademark holder or any of its products. Compliance does not require use of the trademark. Use of the trademark requires permission of the trademark holder.

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.)

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 Life-cycle

4.7.1 General

The UIP state machine, outlined in IEC 62769-2, 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 (3rd 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 IEC 62769-6.

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.

4.7.4 Backward compatibility

For UIPs built for an older FDI® Technology Version the UIP host shall have a binding redirect in place to execute the UIP using the newer FDI® Type Library (see Figure 4).

```
<assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
  <dependentAssembly>
    <assemblyIdentity name="Fdi" publicKeyToken="a591af567772af2f" />
    <bindingRedirect oldVersion="0.0.0.0-1.1.65535.65535" newVersion="1.2.0.0" />
  </dependentAssembly>
</assemblyBinding>
```

IEC

Figure 4 – Example snippet of a UIP host config file for the binding redirect

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
- specific UI Actions (UI actions, which are specific to one UIP, e.g. Reset, Reconnect).

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 5 shows an example of an IAsyncPattern based Asynchronous service execution.

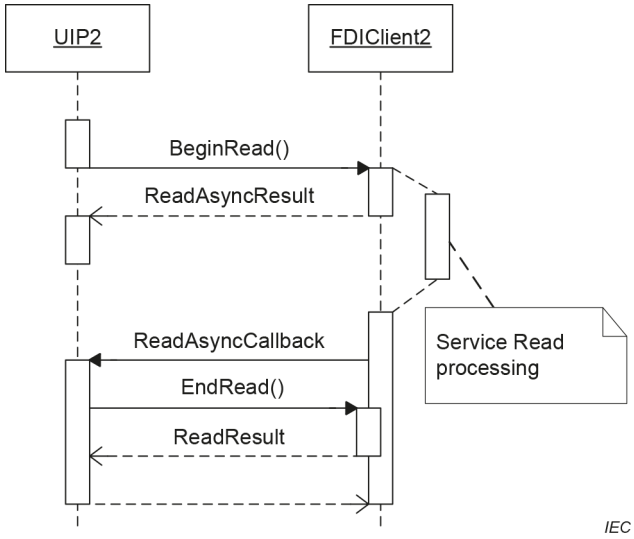


Figure 5 – IAsyncPattern based asynchronous service execution example

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 6.

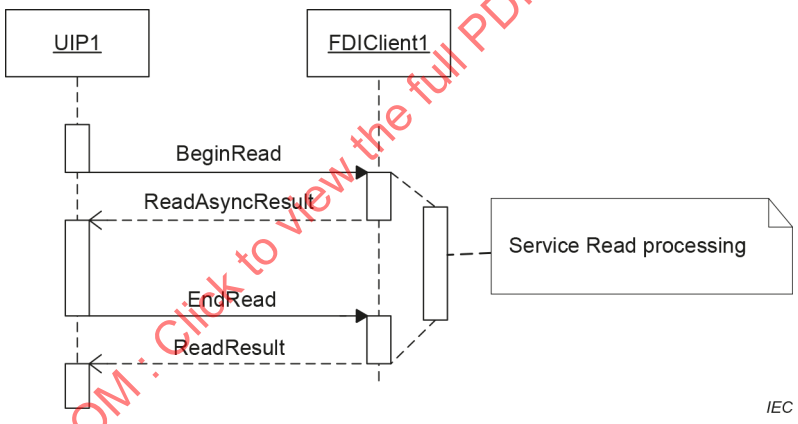


Figure 6 – Blocking service execution example using IAsyncResult based pattern

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

4.8.4 Cancel service execution

Some services specified for the interface `IDeviceModel` (see Table 2) support canceling a started service by means of the function Cancel*OperationName*. The following Figure 7 will illustrate the processing sequence based on the Read service example.

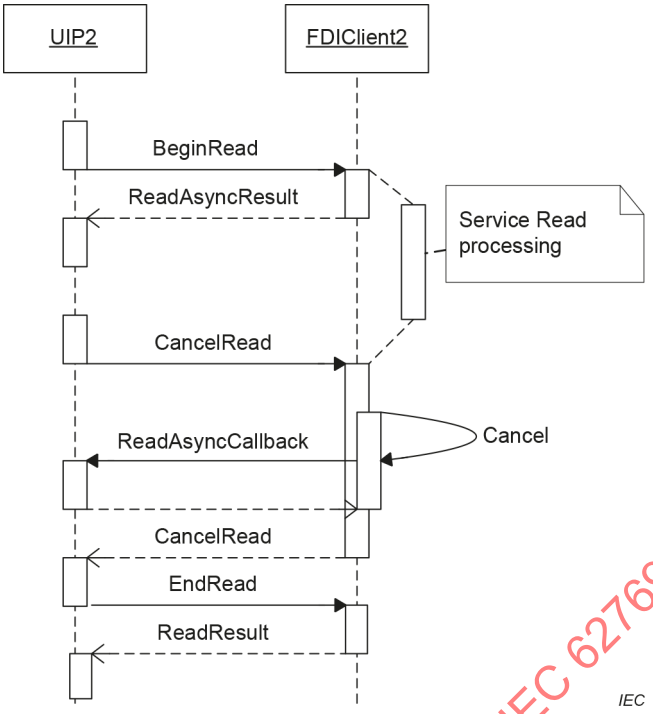


Figure 7 – Cancel service processing sequence example

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 FDI®Client2). 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 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 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.

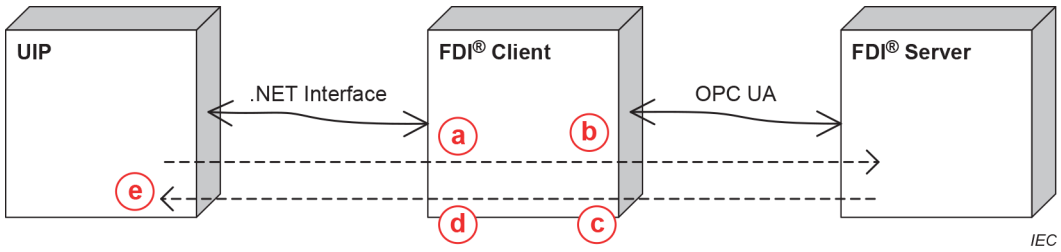


Figure 8 – Exception source

According to the FDI® Architecture exceptions can occur in different steps of the service processing, see Figure 8:

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. Vice versa a UIP implements the accessibility function. The related rules 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. Vice versa a UIP implements the accessibility function. The related rules 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 stability, integrity and sensitive data.

System wide security begins with the design process, which is out of the standardization scope. From the system 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 compliance testing and certification the presumption is that such certified FDI® Packages don't pose any threats to a system. This means a UIP could be executed with full trusted permissions.

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

4.9.2 Access permissions

4.9.2.1 General

Within this section 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 according to this list.

4.9.2.3 Implementation rules

To restrict permissions of an 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 UIP modal or start a new modal UIP.

4.9.3 Code identity concept

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

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

– Unique identification of UIP executables.
– Code integrity verification.

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

5 Interface definition

The following tables specify the mapping between the abstract services specified in IEC 62769-2 and the corresponding .NET implementation which is found in the FDI® Type Library file "fdi.nupkg" and a related help file "FDI Interfaces and Data Types.chm".

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

Table 1 specifies the mapping of the Base Property Services.

Table 1 – Base Property Services

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

Table 2 specifies the mapping of the Device Model Services.

Table 2 – 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 3 specifies the mapping of the Access Control Services.

Table 3 – Access Control Services

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

Table 4 specifies the mapping of the Direct Access Services.

Table 4 – Direct Access Services

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

Table 5 specifies the mapping of the Hosting Services.

Table 5 – 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
ExportFile	
CancelExportFile	
ImportFile	
CancelImportFile	
OpenDefaultApplication	
GetEnvironmentProperties	

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 6 specifies the mapping of the UIP Services.

Table 6 – 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
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 7 specifies the mapping of the base data types.

Table 7 – 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 8 specifies the mapping of the special data types.

Table 8 – 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® Type Library in fdi.nupkg (.NET Assembly)
– FDI® Interfaces and Data Types.chm (Help File)

Bibliography

IEC 61804 (all parts), *Devices and integration in enterprise systems – Function blocks (FB) for process control and electronic device description language (EDDL)*

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

FCG TS10099, *Field Device Integration (FDI®) – Technology Management*

SOMMAIRE

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

_____

INTÉGRATION DES APPAREILS DE TERRAIN (FDI®) –

Partie 6-100: Mapping de technologies – .NET

AVANT-PROPOS

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

L'IEC 62769-6-100 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. Il s'agit d'une Norme internationale.

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

Projet	Rapport de vote
65E/868/CDV	65E/925/RVC

Le rapport de vote indiqué dans le tableau ci-dessus donne toute information sur le vote ayant abouti à son approbation.

La langue employée pour l'élaboration de cette Norme internationale est l'anglais.

Ce document a été rédigé selon les Directives ISO/IEC, Partie 2, il a été développé selon les Directives ISO/IEC, Partie 1 et les Directives ISO/IEC, Supplément IEC, disponibles sous www.iec.ch/members_experts/refdocs. Les principaux types de documents développés par l'IEC sont décrits plus en détail sous www.iec.ch/publications.

Une liste de toutes les parties de la série IEC 62769, publiées sous le titre général *Intégration des appareils de terrain (FDI®)*, se trouve 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 webstore.iec.ch dans les données relatives au document recherché. A 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 ce document indique qu'elle contient des couleurs qui sont considérées comme utiles à une bonne compréhension de son contenu. Les utilisateurs devraient, par conséquent, imprimer ce document en utilisant une imprimante couleur.

INTÉGRATION DES APPAREILS DE TERRAIN (FDI®) –

Partie 6-100: Mapping de technologies – .NET

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®[1], *Field Device Integration*). Le mapping de technologies porte essentiellement sur la mise en œuvre des composants Client FDI® et Plugiciel d'interface utilisateur (UIP, *User Interface Plug-in*) à l'aide de l'environnement d'exécution .NET. Cet environnement d'exécution n'est spécifique qu'à la plateforme WORKSTATION définie dans l'IEC 62769-4.

2 Références normatives

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

IEC 62769-1:2021, *Intégration des appareils de terrain (FDI®) – Partie 1: Vue d'ensemble*

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

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

IEC 62769-6, *Intégration des appareils de terrain (FDI®) – Partie 6: Mapping de technologies*

3 Termes, définitions, abréviations, acronymes et symboles

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

---

[1] FDI est une marque déposée de l'organisation à but non lucratif Fieldbus Foundation, Inc. Cette information est donnée à l'intention des utilisateurs du présent document et ne signifie nullement que l'IEC approuve le détenteur de la marque ou l'emploi de ses produits. La conformité n'exige pas l'utilisation de la marque. L'utilisation de la marque exige l'autorisation du détenteur de la marque.

3.1.2
Assemblage
bloc de construction réutilisable, autodescriptif et qui fournit des informations de version d'une application CLR (*Common Language Runtime*)

3.1.3
Bibliothèque de types FDI®
fichier TypeScript qui contient les interfaces et les types de données utilisés pour l'échange de données et l'interaction entre un UIP et un Client FDI®

3.1.4
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

3.1.5
Registre Windows
base de données définie par le système dans laquelle les applications et les composants systèmes stockent et récupèrent les données de configuration

3.2 Abréviations et acronymes

Pour les besoins du présent document, les abréviations et acronymes de l'IEC 62769-1 et l'IEC 62769-6 ainsi que les suivants s'appliquent.

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).

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 la base technologique de la mise en œuvre du Plugiciel d'interface utilisateur (UIP) qui repose sur l'environnement d'exécution .NET Framework CLR4, ainsi que 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 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 l'achèvement de l'exécutable de l'UIP sont décrits en 4.7. Le 4.8 définit les règles d'interaction entre le 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 Bibliothèque de types FDI®

Les Services d'accès à l'Appareil et les Services d'UIP peuvent être modélisés comme des interfaces .NET qui transmettent des 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®. Afin de gérer les erreurs d'exécution durant les appels de méthode d'interface, des classes d'exception .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 fournie dans un Paquetage Nuget, qui contient un ou plusieurs Assemblages de noms forts. Le nom de fichier de ce Paquetage Nuget doit être Fdi.<version>.nupkg. La Bibliothèque de types FDI® doit être versionnée conformément à l'IEC 62769-1:2021, 8.1. La Bibliothèque de types FDI® fait partie de la Technologie FDI® de base, conformément à l'IEC 62769-1:2021, 8.3.2.1. Par conséquent, elle a une incidence directe sur la Version de Technologie FDI®. Toutes les modifications compatibles de la Bibliothèque de types FDI® se traduisent par une incrémentation de la partie mineure de la Version de Technologie FDI®. Les modifications incompatibles se traduisent par une augmentation de la partie majeure de la Version de Technologie FDI® (voir l'IEC 62769-1:2021, 8.3.2.2). Les informations de version de la Bibliothèque de types FDI® sont disponibles dans la FCG TS10099.

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®. Les Plugiciels d'interface utilisateur (UIP) et l'Application du Client FDI® doivent utiliser cette instance de la Bibliothèque de types FDI®. 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 la Bibliothèque de types FDI®.

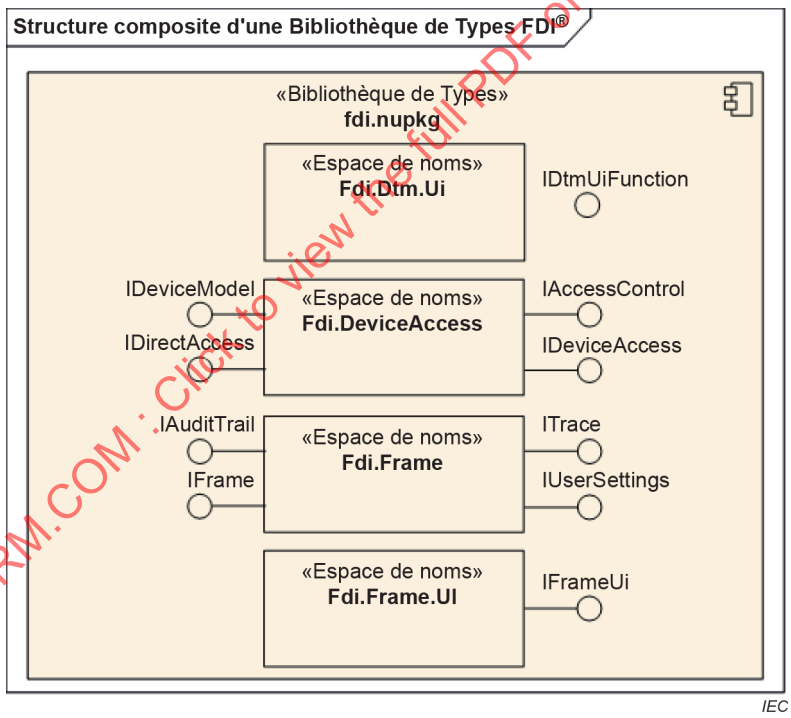


Figure 1 – Structure de la Bibliothèque de types FDI®

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

4.2 Représentation de l'UIP

La Variante d'UIP peut contenir un seul ou plusieurs modules d'exécution (Assemblages .NET) et leurs fichiers supplémentaires connexes (fichiers de ressources, par exemple). Le module d'exécution de la Variante d'UIP est appelé exécutable de l'UIP. Le ou les fichiers supplémentaires de la Variante d'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 Des exemples de fichiers de données supplémentaires de l'UIP comprennent les fichiers de ressources et les données de configuration d'application.

Les identificateurs d'exécution (RuntimeIds) et les versions .NET Framework pris en charge pour une Version de Technologie FDI® particulière sont spécifiés dans la FCG TS10099, "FDI® Technology Management".

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

4.3 Représentation des exécutables de l'UIP

La mise en œuvre de l'UIP dépend du type des 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 WPF `System.Windows.Controls`, classe `UserControl`.

Les exécutables de l'UIP et leurs bibliothèques exigées doivent avoir des noms forts. La signature par nom fort d'un Assemblage peut être réalisée à l'aide d'une clé autogénérée.

NOTE L'identité des Assemblages par nom fort est constituée d'un nom, d'une version, d'un environnement, d'un jeton de clé publique et d'une signature numérique.

Les exécutables de l'UIP et leurs bibliothèques exigées doivent être livrés avec le fichier qui contient la clé publique afin de permettre la vérification de l'Assemblage.

4.4 Règles de compatibilité des exécutables de l'UIP

Les règles de compatibilité pour les différentes versions du composant d'UIP sont spécifiées dans l'IEC 62769-4.

La plateforme cible de compilation pour l'UIP doit être "anyCPU". Si cette plateforme n'est pas possible, l'UIP doit être livré en deux variantes. Une variante d'UIP doit être compilée pour la plateforme cible "x86". La deuxième variante d'UIP doit être compilée pour la plateforme cible "x64". La plateforme cible de compilation doit être décrite dans le fichier catalog.xml qui est défini dans l'IEC 62769-4. Ce fichier catalog.xml contient un élément XML "CpuInformation" qui décrit la variante du Plugiciel d'interface utilisateur. Les valeurs admises à utiliser dans l'élément XML "CpuInformation" sont "anyCPU", "x86" ou "x64".

4.5 Versions .NET CLR admises

4.5.1 Généralités

Des versions CLR spécifiques sont publiées pour l'exécution des composants logiciels conçus avec des versions .NET Framework spécifiques. La version .NET 4.0 du CLR est utilisée pour exécuter des composants logiciels intégrés avec .NET Framework 4.0. Les composants .NET ne sont construits que pour une seule version CLR, mais peuvent être en mesure de fonctionner aussi sous une version CLR plus récente.

Les Clients FDI® peuvent être construits selon la version CLR 4.0 ou une version ultérieure. Un Client FDI® doit se rendre compte des conditions suivantes lors du démarrage d'un UIP.

- Lorsque l'UIP à démarrer a été construit pour la même exécution, l'UIP peut être démarré dans le Client FDI® comme habituellement.

- Lorsque l'UIP à démarrer a été construit avec une autre version CLR et qu'il n'est pas compilé pour la version exécutable actuelle de CLR, le Client FDI® doit démarrer l'UIP dans un processus de substitution avec une version CLR appropriée. (Des informations plus détaillées sont fournies en 4.5.2.)

Compte tenu de ce comportement, un UIP doit être développé pour la version CLR 4.0 ou une version ultérieure. Lorsque les versions CLR ne sont pas compatibles, l'UIP doit être démarré dans un processus séparé. L'UIP n'apparaît alors pas comme un module intégré au sein du Client FDI®. Il incombe au Client FDI® de réaliser le processus de substitution.

4.5.2 Stratégie de compatibilité du CLR

Dans le futur, les Clients FDI® et les UIP pourront être construits sur différentes versions incompatibles du CLR.

Lorsqu'un Client FDI® détecte qu'un UIP exige un CLR qui n'est pas compatible avec le Client FDI®, le Client FDI® peut utiliser une classe "proxy" qui permet une interaction avec l'UIP construit, au moyen d'une version différente du CLR.

Le Client FDI® charge un exécutable de l'UIP du proxy, crée une instance de la classe "proxy" et délègue l'exécution de l'UIP à ce proxy. Le proxy démarre un processus avec le CLR exigé et exécute l'UIP dans ce processus de substitution. Les classes "proxy" fournissent les interfaces FDI® normalisées. Le Client FDI® peut utiliser ces interfaces pour interagir avec l'UIP exécuté dans le processus de substitution.

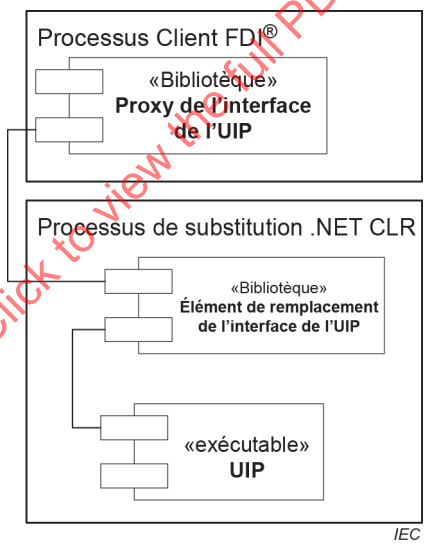


Figure 2 – Processus de substitution .NET

4.5.3 Identification de la plateforme cible .NET d'un UIP

L'information relative à la version CLR de la plateforme cible .NET pour laquelle est compilé un certain Assemblage peut être extraite au moyen des fonctions de la bibliothèque .NET Framework (voir Figure 3).

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

IEC

Figure 3 – Identification de la version d'exécution

NOTE L'environnement de développement Intégré (IDE, *Integrated Development Environment*) Visual Studio®[2] 2008 et 2010 permet aux développeurs de choisir la cible .NET Framework. Le choix d'une cible .NET Framework plus ancienne que la base relative à la version actuelle de l'IDE Visual Studio® crée automatiquement un fichier de configuration répertorié comme "app.config" dans l'explorateur de la solution. Ce fichier reflète seulement les paramètres du compilateur actuel. Le compilateur ne lit pas ce fichier.

4.6 Déploiement de l'UIP

Les règles générales d'installation de l'UIP sont décrites dans l'IEC 62769-2. L'exécutable de l'UIP ne doit pas être enregistré dans le Cache global d'assemblages.

La règle "nom fort" permet de s'assurer que les Assemblages liés de différentes versions de l'UIP peuvent coexister durant l'exécution.

La mise en œuvre du Client FDI® permet de s'assurer que le déploiement de l'UIP fonctionne indépendamment des informations d'identification de l'utilisateur actuel (voir la NOTE ci-dessous).

NOTE Certains dossiers gérés par le système d'exploitation exigent des droits d'accès spécifiques, par exemple, des modifications dans le dossier "Programmes" exigent des droits "Administrateur". Le système d'exploitation Windows fournit plusieurs moyens qui permettent à une application en cours d'exécution avec des droits utilisateur limités d'exécuter des actions avec des privilèges administrateur transparents pour l'utilisateur, par exemple, traitement de restrictions particulières pour des répertoires identifiés, services avec des droits d'administration, exécutables qui sont configurés pour s'exécuter automatiquement avec des droits d'administration. La solution alternative consiste à copier les exécutables de l'UIP dans des dossiers inscriptibles pour les utilisateurs "ordinaires".

4.7 Cycle de vie de l'UIP

4.7.1 Généralités

Le diagramme d'états de l'UIP, décrit dans l'IEC 62769-2, est composé des états Loaded, Created, Operational, Deactivated et Disposed. Les mécanismes qui ont une incidence sur les changements d'états sont décrits en 4.7.

Lorsque le Client FDI® a stocké l'exécutable de l'UIP sur le Client FDI®, le Client FDI® charge les Assemblages UIP dynamiquement dans la mémoire et exécute la logique connexe par l'appel des fonctions correspondantes de l'interface FDI® spécifiée.

Le 4.7 décrit les règles qui définissent comment le Client FDI® doit activer et désactiver l'UIP.

4.7.2 Etapes d'activation de l'Assemblage UIP

4.7.2.1 Load

Le Client FDI® doit charger les exécutables de l'UIP au moyen du mécanisme LoadFrom. Pour ce faire, .NET Framework fournit le System.Reflection.Assembly.LoadFrom.

Le mécanisme LoadFrom se comporte comme suit:

- LoadFrom charge l'Assemblage adressé avec le chemin du fichier et aussi les Assemblages référencés situés dans le même répertoire. La chaîne d'argument assemblyFile doit contenir le nom du fichier de l'exécutable de l'UIP. Le nom du fichier de l'exécutable de l'UIP représente la fonction StartElementName décrite dans l'IEC 62769-4.
- Si un Assemblage est chargé avec LoadFrom, et que plus tard un Assemblage dans le "contexte de chargement" tente de charger le même Assemblage par le nom d'affichage, alors cette tentative de chargement échoue.

---

2 Visual Studio est une marque de Microsoft Corporation. Cette information est donnée à l'intention des utilisateurs du présent document et ne signifie nullement que l'IEC approuve le détenteur de la marque ou l'emploi de ses produits. La conformité n'exige pas l'utilisation de la marque. L'utilisation de la marque exige l'autorisation du détenteur de la marque.

- Si un Assemblage avec la même identité est déjà chargé (par exemple, par un autre UIP), alors LoadFrom renvoie l'Assemblage qui a été chargé précédemment même si un chemin de fichier différent a été spécifié. Même l'emploi d'un nom de fichier différent n'a pas d'incidence. Seule l'identité de l'Assemblage est importante.
- Si un Assemblage est chargé avec LoadFrom et que le chemin de vérification inclut un Assemblage avec la même identité (par exemple, dans le Cache global d'assemblages ou dans un répertoire de l'application), cet Assemblage est alors chargé même si un chemin de fichier différent a été spécifié.
- `LoadFrom` exige les autorisations `FileIOPermissionAccess.Read` et `FileIOPermissionAccess.PathDiscovery` ou `WebPermission` sur le chemin spécifié.
- `LoadFrom` charge l'assemblage dans le Domaine d'Application par défaut.
- Lorsqu'une image native d'un Assemblage (générée par ngen.exe) existe pour le chemin de fichier spécifié, elle n'est alors pas utilisée. L'Assemblage ne peut pas être chargé comme indépendant du domaine, c'est-à-dire que l'Assemblage ne peut pas être partagé entre les Domaines d'Application.

Ce comportement impose les règles de déploiement suivantes:

- règles relatives aux dépendances des Assemblages (voir 4.7.2.4.2).

Le Client FDI® ne doit utiliser que `LoadFrom`. L'utilisation d'autres moyens de chargement d'Assemblages .NET/de création d'objets n'est pas admise.

- règles relatives aux Assemblages partagés (voir 4.7.2.4.3);
- un code machine spécifique d'un processeur précompilé ne peut pas être utilisé;
- les aspects de sécurité qui concernent le chargement et l'exécution des Assemblages sont décrits en 4.9.

4.7.2.2 Create (créer)

La création d'une instance de l'Assemblage UIP fonctionne en utilisant les fonctions de bibliothèque .NET `System.Reflection.Assembly.GetTypes` et `System.Activator.CreateInstance`. La bibliothèque de types FDI® déclare un "attribut personnalisé" dénommé `UIPActivationClass`. Cet attribut ne doit être ajouté qu'à l'objet qui met en œuvre l'interface `IDtmUiFunction` qui met réellement en œuvre la fonction de démarrage de l'UIP. L'attribut `UIPActivationClass` doit être utilisé seulement une fois.

Le Client FDI® peut désormais utiliser les services `System.Reflection` afin de déterminer clairement la procédure d'activation mise en œuvre de l'UIP.

NOTE 1 La fonction System.Reflection.Assembly.GetTypes peut être utilisée pour interroger l'interface IDtmUiFunction.

NOTE 2 La fonction System.Attribute.GetCustomAttributes peut être utilisée pour lire les attributs personnalisés supplémentaires.

NOTE 3 Le résultat de l'appel de fonction System.Activator.CreateInstance est un objet de type IDtmUiFunction.

Une diffusion des types de données est nécessaire.

4.7.2.3 Activate (Activer)

L'appel de la fonction `IDtmUiFunction.Init` active finalement l'UIP pour l'utilisateur.

4.7.2.4 Bibliothèques externes

4.7.2.4.1 Généralités

Les Assemblages UIP peuvent s'appuyer sur des bibliothèques externes (bibliothèques tierces) et d'autres Assemblages, comme des bibliothèques de contrôle d'utilisateur spécifiques. Les Clients FDI® n'effectuent pas d'installation d'UIP, mais ils chargent dynamiquement et exécutent l'UIP. Pour prendre en charge cette utilisation, ainsi que l'exigence de prévenir les problèmes éventuels des Assemblages en conflit, des règles sont spécifiées pour les bibliothèques externes.

Les bibliothèques externes:

- doivent être contenues dans le Paquetage FDI®;
- ne doivent pas exiger l'installation de Microsoft Installer (MSI);
- ne doivent pas exiger d'entrées dans le Registre Windows ou dans le Cache global d'assemblages;
- doivent respecter les restrictions d'accès décrites en 4.9.2;
- doivent être compatibles avec les plateformes décrites dans l'IEC 62769-6.

4.7.2.4.2 Chargement de librairies externes

Le Client FDI® charge l'Assemblage UIP qui contient la classe "main" de l'UIP de mise en œuvre de l'interface `IDtmUiFunction`, par appel de la fonction `LoadFrom` du .NET Framework. Les Assemblages référencés qui sont stockés dans le même répertoire sont chargés automatiquement avec cet Assemblage .NET. Les Assemblages référencés qui sont stockés à d'autres emplacements (par exemple dans un sous-répertoire) doivent être chargés explicitement par l'UIP lui-même.

L'UIP doit charger ces Assemblages également en appelant la fonction `LoadFrom` du .NET Framework. Le chargement des Assemblages par d'autres méthodes .NET Framework n'est pas admis.

L'utilisation de bibliothèques externes ne doit pas compromettre l'exigence d'autonomie pour les Paquetages FDI®; toutes les bibliothèques externes doivent être incluses dans le Paquetage FDI® de l'UIP.

4.7.2.4.3 Chargement de bibliothèques externes partagées

Une bibliothèque externe est une bibliothèque externe partagée si l'identité d'un Assemblage .NET connexe peut être utilisée par différents exécutables de l'UIP. L'identité d'un Assemblage .NET est importante. Le chemin d'installation et le nom du fichier de l'Assemblage ne sont pas pertinents.

L'utilisation de bibliothèques partagées ne doit pas compromettre l'exigence d'autonomie pour les Paquetages FDI®. Chaque Paquetage FDI® fourni doit être livré avec toutes les bibliothèques UIP connexes exigées. Le mécanisme de partage vient du mécanisme d'optimisation mis en œuvre du .NET Framework.

Lorsqu'un Assemblage partagé est utilisé, les règles suivantes s'appliquent:

- toute modification incompatible de l'Assemblage partagé doit conduire à une nouvelle identité, par exemple un numéro de version différent;
- les Assemblages partagés ne doivent pas prétendre être chargés à partir d'un chemin d'installation spécifique, par exemple en comptant sur le fait que certains fichiers sont stockés dans le même répertoire ou dans un sous-répertoire;

- les variables statiques dans les Assemblages partagés sont également partagées lorsque l'Assemblage est chargé dans le même Domaine d'Application. Ainsi, les variables statiques ne doivent pas avoir d'effets secondaires dans de tels scénarios. Les bibliothèques partagées externes ne doivent pas déclarer de variables statiques.
- en raison de la règle d'autonomie définie pour le Paquetage FDI®, les Assemblages partagés doivent être déployés avec tous les Paquetages FDI® qui utilisent un Assemblage partagé.

4.7.2.5 Appel du constructeur de l'UIP

La mise en œuvre du constructeur et du destructeur ne doit pas générer d'exceptions. La logique du constructeur doit être limitée pour instancier l'objet en ce qui concerne la structure de données internes. La logique du destructeur doit être limitée pour détruire l'objet en ce qui concerne les ressources de mémoire de publication. Le constructeur et le destructeur ne doivent pas:

- envoyer de rappel au Client FDI®;
- appeler d'interaction utilisateur.

4.7.3 Etapes de désactivation de l'Assemblage UIP

4.7.3.1 Deactivate (Désactiver)

Pour désactiver l'UIP, le Client FDI® doit appeler les interfaces `IDtmUiFunction.BeginClose` et `IDtmUiFunction.EndClose`. Après une exécution réussie, l'UIP doit libérer toutes les ressources et le Client FDI® doit supprimer toutes les références vers l'instance de l'UIP. Le nettoyeur .NET supprime enfin l'objet d'exécution d'UIP.

4.7.3.2 Dispose (supprimer)

Un Assemblage .NET qui est chargé dans un processus, respectivement dans le `ApplicationDomain` lié, n'est jamais déchargé, sauf lorsque le `ApplicationDomain` lui-même est détruit. Cette situation signifie que si le Client FDI® charge un Assemblage UIP dans le `ApplicationDomain` par défaut, ces Assemblages et tous les Assemblages dépendants ne sont alors jamais déchargés, sauf si l'application est fermée.

Les Assemblages UIP doivent être développés en tenant compte de ce comportement de .NET Framework. Pour réduire la consommation de mémoire, les règles suivantes s'appliquent:

- réduire le plus possible l'utilisation de variables statiques, car celles-ci augmentent la consommation de mémoire de l'Assemblage;
- déplacer la fonctionnalité de l'UIP qui n'est pas toujours (ou rarement) nécessaire pour séparer les Assemblages. Ces Assemblages ne sont alors chargés automatiquement ou manuellement que lorsque le code correspondant est exécuté;
- utiliser les Assemblages partagés lorsque cela est possible;
- le Client FDI® peut exécuter les Assemblages .NET dans un Domaine d'Application séparé afin d'avoir la capacité de les décharger.

4.7.4 Rétrocompatibilité

Pour les UIP construits pour une Version de Technologie FDI® plus ancienne, une redirection de liaison doit être en place sur l'hôte de l'UIP afin d'exécuter l'UIP à l'aide de la nouvelle Bibliothèque de types FDI® (voir Figure 4).

```
<assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
  <dependentAssembly>
    <assemblyIdentity name="Fdi" publicKeyToken="a591af567772af2f" />
    <bindingRedirect oldVersion="0.0.0.0-1.1.65535.65535" newVersion="1.2.0.0" />
  </dependentAssembly>
</assemblyBinding>
```

IEC

Figure 4 – Exemple d'extrait du fichier de configuration d'un hôte UIP, relatif à la redirection de liaison

4.8 Interaction entre un Client FDI® et un UIP

4.8.1 Traitement des éléments d'UI normalisés

Les UIP doivent déléguer au Client FDI® la présentation et le traitement des éléments d'UI normalisés. Ces éléments d'UI normalisés sont:

- les Actions d'UI avec la sémantique normalisée (Appliquer/Fermer/Aide en ligne); et
- les Actions d'UI spécifiques (Actions spécifiques à un IUP, par exemple Réinitialiser, Reconnecter).

Pour assurer des interactions cohérentes de l'Interface utilisateur dans les UIP de différents fournisseurs, un UIP peut déléguer au Client FDI® la présentation et le traitement d'actions supplémentaires spécifiques à l'UIP. Néanmoins, les UIP peuvent mettre en œuvre des actions d'UI non normalisées dans leur propre partie UI.

L'ensemble des actions d'UI normalisées et de leur sémantique respective est fixe. Cependant, la disponibilité de ces actions peut varier à tout moment en fonction de l'état interne de l'UIP. L'ensemble des actions supplémentaires spécifiques à l'UIP et de leur disponibilité n'est pas fixe. Un UIP peut ajouter, supprimer, renommer, activer ou désactiver les actions spécifiques à l'UIP à tout moment en fonction de ses exigences. L'UIP doit informer le Client FDI® chaque fois que la disponibilité de ses actions normalisées ou des actions spécifiques à l'UIP varie (voir les événements `IStandardActions.StandardActionItemSetChanged` et `IApplicationSpecificActions.ApplicationSpecificActionItemSetChanged`).

Un Client FDI® peut utiliser des éléments d'UI spécifiques, par exemple les contrôles de boutons, pour fournir un accès direct aux actions normalisées, mais également pour les appeler indirectement dans le cadre de l'interaction entre l'utilisateur et d'autres éléments d'interface utilisateur du Client FDI®. Le Client FDI® doit toujours présenter toutes les actions personnalisées exposées par un UIP avec des éléments d'UI spécifiques.

4.8.2 Exécution d'un service sans blocage

4.8.2.1 Fonctions internes du Client FDI®

La mise en œuvre de la fonction Begin*NomOpération* doit copier le contenu de l'Argument `asyncState` dans le membre `AsyncState` de l'objet `IAsyncResult` renvoyé.

La partie productive (chronophage) de la fonction nommée *NomOpération* doit être exécutée dans un fil différent. La synchronisation avec le fil d'appel est traitée par l'objet `AsyncWaitHandle` (classe `WaitHandle`) qui est aussi un membre de l'objet `IAsyncResult`.

Lorsque le traitement de la partie productive de la fonction nommée *NomOpération* est terminé, l'attribut `IsCompleted` des objets `IAsyncResult` doit être défini sur `True`. Lorsque la valeur de l'argument `AsyncCallBack` est valide (n'est pas NULL), le Client FDI® notifie l'UIP par le rappel.

La mise en œuvre de Cancel*NomOpération* utilise l'argument `IAsyncResult` pour identifier le service qui a été démarré avec Begin*NomOpération*. Lorsque Begin*NomOpération* a démarré un service OPC UA, le Client FDI® doit appeler le service Cancel défini par OPC UA.

4.8.2.2 Fonctions internes de l'UIP

La gestion de plusieurs services asynchrones en parallèle doit être assurée à l'aide de l'objet `AsyncState`.

L'objet `IAsyncResult` renvoyé par Begin*NomOpération* contient l'objet `WaitHandle`. L'UIP doit effectuer une synchronisation de son propre fil au moyen de l'objet `WaitHandle`.

4.8.2.3 Séquence d'exécution d'un service sans blocage

Le paragraphe suivant spécifie la séquence d'interaction entre le Client FDI® et l'UIP. Les mécanismes de gestion de fils mis en œuvre à l'intérieur du Client FDI® ne sont pas représentés. L'interaction entre un Client FDI® et un Serveur FDI® repose sur le modèle Demande/Réponse. La demande de service du Client FDI® correspond à la fonction Begin*NomOpération*. L'appel d'AsyncCallback correspond à la réception de la réponse du service Client. End*NomOpération* transmet les résultats contenus dans la réponse. La mise en œuvre de l'exécution d'un service sans blocage n'exige aucune gestion de fils à l'intérieur du Client FDI®. La Figure 5 représente un exemple d'exécution d'un service asynchrone fondé sur IAsyncPattern.

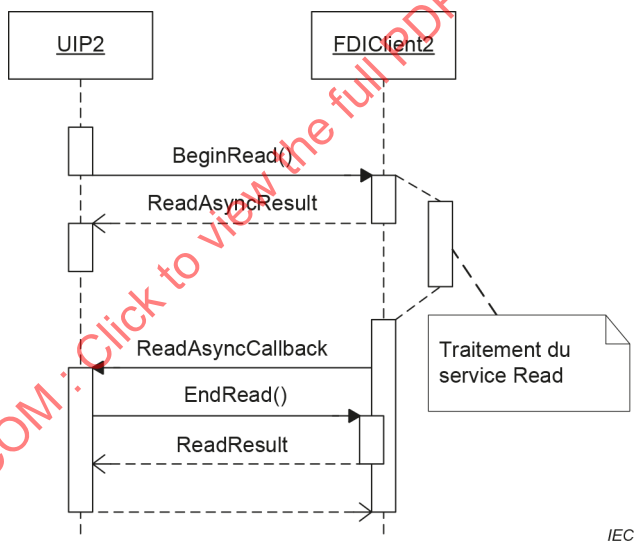


Figure 5 – Exemple d'exécution d'un service asynchrone fondé sur IAsyncPattern

4.8.3 Exécution d'un service de blocage

Les interfaces fournies par le Client FDI® permettent de donner un accès au Modèle d'information synchrone par la fonctionnalité décrite en 4.8.1 de la façon représentée à la Figure 6.

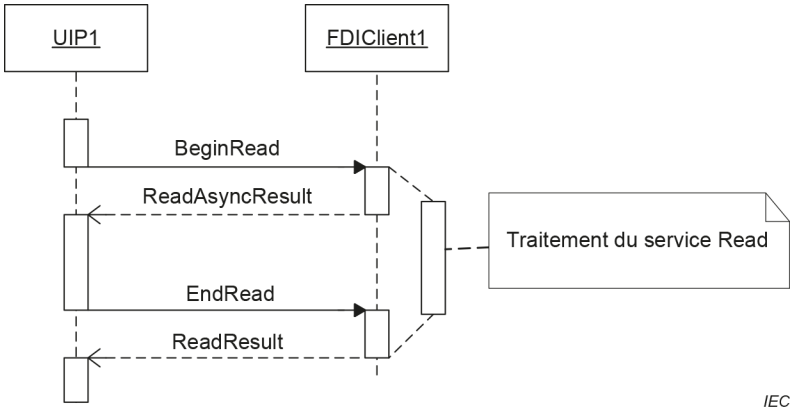


Figure 6 – Exemple d'exécution d'un service de blocage avec le modèle fondé sur IAsyncResult

`ReadAsyncResult` est l'objet qui met en œuvre l'interface `IAsyncResult`.

4.8.4 Exécution du service d'annulation

Certains services spécifiés pour l'interface `IDeviceModel` (voir Tableau 2) prennent en charge l'annulation d'un service débuté au moyen de la fonction Cancel*NomOpération*. La Figure 7 représente un exemple de séquence de traitement pour le service Read.

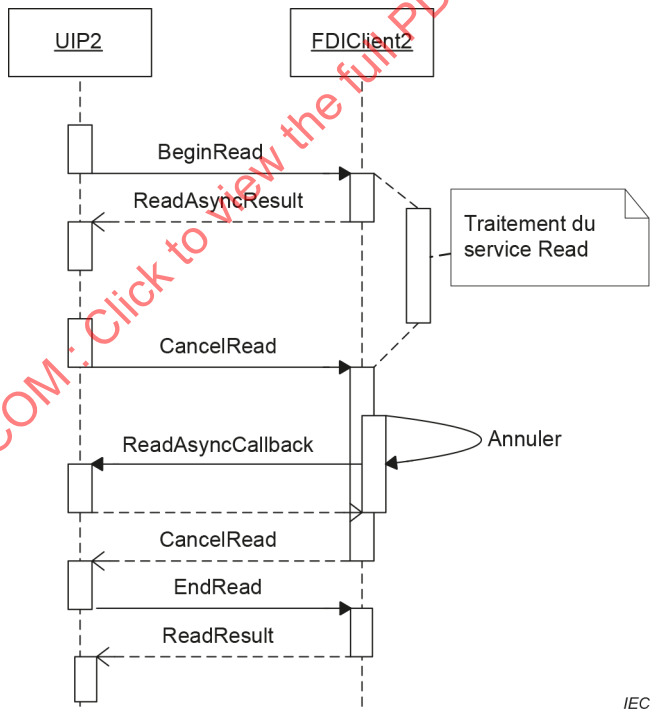


Figure 7 – Exemple de séquence de traitement du service d'annulation

L'appel de `CancelRead` déclenche les fonctions internes du Client FDI® nécessaires pour annuler l'opération de lecture active. Le Client FDI® peut ne pas être en mesure d'annuler immédiatement l'opération, mais il convient qu'il l'annule dès que possible. Lorsque l'opération a été annulée, le Client FDI® notifie l'UIP par `ReadAsyncCallback`. L'UIP doit alors appeler la fonction `EndRead`.

NOTE Un défi général de la mise en œuvre de ce modèle consiste à gérer correctement les situations de compétition des deux côtés (UIP2 et FDI®Client2). Si le Client FDI® a transmis l'exécution du service par un service OPC UA, l'exécution réelle du service se déroule à l'intérieur du Serveur FDI®.

Selon la façon dont l'UIP est hébergé, il peut exister trois processus de travail indépendants. Par conséquent, la demande d'annulation (envoyée par l'UIP) peut être visible dès que le Serveur FDI® a terminé la demande de service. La réponse associée envoyée par le Serveur FDI® peut ne pas avoir été réceptionnée par le Client FDI®. Le Client FDI® peut appeler la fonction ReadAsyncCallBack tandis que l'UIP appelle la fonction CancelRead.

`ReadAsyncResult` est l'objet qui met en œuvre l'interface `IAsyncResult`.

4.8.5 Threading (gestion de fils)

4.8.5.1 Règles de mise en œuvre

L'UIP doit être en mesure de recevoir des appels dans n'importe quel fil.

L'UIP ne doit pas bloquer les appels effectués par le Client FDI®.

L'UIP ne doit pas utiliser le fil du Client FDI® pour signaler en retour le rappel au Client FDI® lui-même. Il s'agit d'éviter les impasses et les boucles infinies.

L'UIP ne doit pas exécuter d'opérations synchrones, comme cela est décrit en 4.8.3, dans le fil de l'Interface utilisateur: le fil de l'Interface utilisateur d'un processus ne doit être destiné qu'à la réception des entrées de l'utilisateur et qu'à l'exécution des tâches de dessin.

L'UIP et le Client FDI® ne doivent pas bloquer le fil de l'Interface utilisateur. L'Interface utilisateur doit toujours rester réactive. Le fil de l'Interface utilisateur est partagé entre les différents objets liés à l'Interface Utilisateur FDI® pour l'entrée utilisateur et les tâches de dessin. Lorsqu'un objet bloque ce fil afin d'effectuer un certain traitement, ce blocage a une incidence sur la réactivité des autres interfaces utilisateur.

L'UIP et le Client FDI® ne doivent pas bloquer un appel de méthode Begin*NomOpération*: une méthode Begin*NomOpération* ne doit démarrer qu'une opération asynchrone. L'appelant ne doit pas être bloqué.

4.8.6 Délais d'expiration

Les interfaces référencées à l'Article 5 permettent d'exécuter des services asynchrones. Le temps d'exécution de ces services dépend de contraintes de performance liées à la communication des bus ou aux performances du Client FDI®/Serveur FDI®. Les règles énumérées ci-dessous établissent l'interopérabilité du système en ce qui concerne la prévention des "situations de compétition". La règle générale est que le composant peut gérer le traitement des délais d'expiration seulement pour les processus qui sont complètement sous le contrôle de ce composant. La liste suivante indique les éléments de l'ensemble du système qui peuvent mettre en œuvre la fonction de détection des délais d'expiration.

- UIP: il ne doit pas mettre en œuvre la détection des délais d'expiration.
- Logique applicative: elle ne doit pas mettre en œuvre la détection des délais d'expiration (Paquetage FDI®).
- Client FDI®: il doit mettre en œuvre la détection des délais d'expiration. Dans le cas d'une architecture OPC UA, la prise en charge connexe est intégrée dans les piles de communication OPC UA. Les délais d'expiration détectés pendant les opérations effectuées pour le compte de l'UIP doivent être transmis comme des codes de résultat négatifs de la fonction.