IVI Driver .NET Specification

Version Number	Date of Version	Version Notes
1.0	January 21, 2025	Initial Specification Version

Abstract

This specification contains the Microsoft .NET 6+ specific requirements for an IVI.NET driver, it is an IVI Language-Specific specification. Drivers that comply with this specification are also required to comply with the *IVI Driver Core Specification*.

Authorship

This specification is developed by member companies of the IVI Foundation. Feedback is encouraged. To view the list of member vendors or provide feedback, please visit the IVI Foundation website at www.ivifoundation.org.

Table of Contents

Overview of the IVI.NET Driver Language Specification

This specification contains the Microsoft .NET 6+ specific requirements for an IVI.NET driver, it is an IVI Language-Specific specification. Drivers that comply with this specification are also required to comply with the *IVI Driver Core Specification*.

Substitutions

This specification uses paired angle brackets to indicate that the text between the brackets is not the actual text to use, but instead indicates the text that is used in place of the bracketed text. The *IVI Driver Core Specification* describes these substitutions.

IVI.NET Driver Architecture

This section describes how IVI.NET instrument drivers use .NET technology. This section does not attempt to describe the technical features of .NET, except where necessary to explain a particular IVI.NET feature. This section assumes that the reader is familiar with .NET technology.

Operating Systems and Bitness

IVI.NET drivers work on one or more of the following Microsoft operating systems: Windows 10 (64-bit) or Windows 11.

IVI.NET drivers may be provided as 32-bit drivers, 64-bit drivers, or both. Users need to acquire drivers with the correct bitness for their application needs. The compliance document for an IVI driver states whether the driver is available in a 32-bit version, a 64-bit version, or both.

Target .NET Versions

IVI.NET drivers shall target .NET 6 or later. IVI.NET Framework drivers should target .NET Framework 4.6.2 or later. Drivers may be dual targeted.

All references to .NET in this document unless specifically qualified that they refer to .NET Framework refer to .NET 6 or later.

Drivers shall indicate the .NET and .NET FW versions they support in the IVI Compliance document.

IVI.NET Namespaces

The namespace of IVI.NET instrument drivers shall be `<CompanyName>.<DriverIdentifier>` or `<CompanyName>.<Technology>.<DriverIdentifier>`, where:

- `<CompanyName>` is the name of the driver vendor.
- `<Technology>` is an optional field determined by The driver vendor, not by the IVI Foundation.
- `<DriverIdentifier>` is a string that uniquely identifies the driver. `<Identifier>` is returned by the required property of the same name.

For example, `Agilent.Agilent34411` or `NationalInstruments.ModularInstruments.NIDmm` are valid namespaces for IVI.NET drivers.

IVI.NET Driver Classes

IVI.NET instrument drivers shall consist of one or more .NET classes.

One class in the driver assembly, called the *main class:*

- Shall be named `<DriverIdentifier>.*`
- Shall implement the required constructor and interface that are specified by this specification.
- Shall implicitly implement `IDisposable` and shall call Dispose when the driver object is destroyed.
- May implicitly implement `IServiceProvider` if needed to navigate between interfaces that are not part of the driver hierarchy.
- May implement additional interfaces at the developer's discretion.

The driver may (and typically will) define additional classes as needed to fill out the driver's functionality. For example, additional classes are needed to implement repeated capability collections.

IVI.NET Hierarchy

An IVI.NET driver shall organize the driver's API as a hierarchy of interfaces and classes. Each of the interfaces is implemented by one of the driver's classes. Classes may implement zero, one, or more than one interface.

The root of the hierarchy shall be the main class `<DriverIdentifier>.*`.

The main class shall include properties that return references to child interfaces or classes. A child interface or class may in turn include properties that return references to its child interfaces or classes, and so on. These *reference properties* may then be used to navigate to any instrument functionality from the main class. The hierarchy may be arbitrarily deep.

Consider the following example code:

```
Kt1234.Itf2.Cls3.Measure();
```

`Kt1234` is a reference to an instance of the main class. `Kt1234` contains an interface reference property named `Itf2`, which returns a reference to an interface `IItf2`. `IItf2` contains a reference property named `Cls3`, which returns a reference to a class `Cls3`. `Cls3` contains the method `Measure()`.

> **Observation:** > As the user types each of these names, IntelliSense makes navigating the hierarchy easy. It displays a dropdown list of methods and

properties in the corresponding class or interface. After typing `KT1234` followed by a period, a list of all the properties and methods in `Kt1234` appears, allowing the user to select one. After selecting `Itf2` and typing the period, a list of the methods and properties in `Itf2` appears. After selecting `Cls3` and typing the period, a list of the methods and properties in `Cls3` appears, and the user can see and select `Measure()`.

Interface Naming Interfaces defined for IVI.NET drivers should begin with `I` followed by one or more words that describe the interface. If the IVI.NET driver has a root interface, it shall be named `I<DriverIdentifier>`.

Drivers should include the prefix `I<DriverIdentifier>` on driver interface names. For instance a *Trigger* interface should be named `I<DriverIdentifier>Trigger`. This convention avoids interface name conflicts if the driver implements externally defined interfaces.

Reference Property Naming The names of reference properties should be the trailing words of the type of interface the property returns.

For interfaces, the `I<DriverIdentifier>` is omitted.

For example:

```
IKt1234DmmTrigger Trigger { get; }
```

Repeated Capabilities

Repeated capabilities may be represented in two ways in IVI .NET drivers. Repeated capability instances may be specified by a method that selects the active instance (the *selector style*) or by selecting a particular instance from an IVI .NET collection (the *collection style*). See the *IVI Core Driver Specification* for details.

For IVI.NET drivers, collection style repeated capabilities are recommended.

Collection Style Repeated Capabilities and the Hierarchy Collection style repeated capabilities consist of at least two interfaces or classes. The first is the collection itself, and the second is an item in the collection. In the hierarchy, a reference property returns the collection interface or class. Then the collection's Item operator `[]` is used to return an item from the collection. Each item in the collection represents one instance of the repeated capability.

Collection style repeated capabilities may be indexed by a string, integer, or other .NET object.

Consider the following example code:

```
Kt1234.Trace["B"].Peak
```

`Kt1234` is a reference to the main class. `Kt1234` contains an interface reference property named `Trace`, which returns a reference to the Trace collection. The `["B"]` item operator selects the item named "B" from the collection and returns a reference to an interface or class that uniquely represents the "B" trace. That interface or class contains the property `Peak`.

Collections may be implemented in a variety of ways.

- Many collections do not need to add or remove members after they are constructed. These can be implemented directly as .NET read-only collections, preferably dictionaries.
- Collections that do need to add or remove members after they are constructed can be implemented directly as .NET writeable collections, but this carries the risk that clients will add or delete members that they shouldn't. Developers concerned about this risk may wrap the collection and hide features or provide appropriate methods to manipulate the collection.
- Developers may create a custom collection that derives from `IEnumerable`, and implement only the features that are needed.

Repeated Capability Reference Property Naming Drivers should name the classes and interfaces associated with Repeated Capability Reference Properties as described in this section.

In the following `<RcName>` is the name of the repeated capability.

- Repeated capability collection interfaces should be named: `I<DriverIdentifier><RcName>C`
- The interface or class returned by the collection's Item operator should be named: `I<DriverIdentifier><RcName>`
- The interface or class returned by the collection's Item operator should include a property called *Name*. *Name* returns the physical repeated capability identifier defined by the specific driver for the repeated capability that corresponds to the index that the user specifies. If the driver defines a qualified repeated capability name, this property returns the qualified name.

For example, consider a Trigger repeated capability on an Acme 123A instrument:

```
interface IAcme123ATrigger {
  String Name;
```

```
  // ...
}

interface IAcme123ATriggerCollection {
  IAcme123ATrigger operator[](int index) {};
  // ...
}
```

IVI.NET Error Handling

All IVI.NET instrument drivers shall consistently use the standard .NET exception mechanism to report errors. Neither return values nor *out* parameters shall be used to return error information.

> **Observation:** > The IVI method `QueryInstrumentError()` is used to handle errors within the instrument that may not be thrown as .NET exceptions.

Documentation and Source Code

This specification does not have specific requirements on the format or distribution method of documentation and source code other than those called out in *IVI Driver Core Specification*.

> **Observation:** > Driver developers are discouraged from including full documentation and source code in the driver NuGet package. This is because numerous occurrences of the NuGet package could be present on the target system.

Base IVI.NET API

This section gives a complete description of each constructor, method, or property required for an IVI.NET Core driver. The following table shows the mapping between the required base driver APIs described in the IVI Driver Core specification and the corresponding IVI.NET specific APIs described in this section.

Required Driver API Mapping Table

Required Driver API (IVI Driver Core)	Core IVI.NET API
Initialization	Core Driver Constructors
Driver Version	Property: ComponentVersion
Driver Vendor	Property: ComponentVendor

Required Driver API (IVI Driver Core)	Core IVI.NET API
Driver Setup	Property: DriverSetup
Error Query	Method: ErrorQuery()
Instrument Manufacturer	Property:InstrumentManufacturer
Instrument Model	Property: InstrumentModel
Query Instrument Status Enabled	Property:QueryInstrumentStatus
Reset	Method: Reset()
Simulate Enabled	Property: Simulate
Supported Instrument Models	Method: GetSupportedInstrumentModels()

Observation: > The properties *ComponentVersion* and *ComponentVendor* are so named for backwards compatibility, drivers may also include more intuitively named properties such as DriverVersion and DriverVendor.

Constructors

In IVI.NET, constructors provide the initialization functionality described in *IVI Driver Core Specification*. This section specifies the required IVI.NET specific driver constructors.

.NET Constructor Prototypes The IVI.NET drivers shall implement two constructors with the following prototypes.

```
<DriverClassName>(String resourceName, Boolean idQuery, Boolean reset)
```

IVI.NET drivers shall implement an additional constructor that provides a way for the client to specify driver options (such as simulation). The mechanism by which these parameters are passed is driver-specific.

IVI.NET drivers may implement additional constructors.

The parameters are defined in the *IVI Driver Core Specification*. The following table shows their names and types for .NET:

Inputs	Description	Data Type
resourceName	Resource Name	String
idQuery	ID Query	Boolean
reset	Reset	Boolean

Notes:

- *IVI Driver .NET* constructors are implemented in the namespace of the specific driver, on the main driver class.

IIviDriverCore Interface

```
namespace "Ivi.DriverCore"

public struct ErrorQueryResult
{
    Int32 Code { get; }
    String Message { get; }
}

public interface IIviDriverCore
{
    String DriverVersion { get; }
    String DriverVendor { get; }
    String DriverSetup { get; }
    String InstrumentManufacturer { get; }
    String InstrumentModel { get; }
    Boolean QueryInstrumentStatus { get; set; }
    Boolean Simulate { get; }
    ErrorQueryResult ErrorQuery();
    void Reset();
    String[] GetSupportInstrumentModels();
}
```

IVI publishes the interface shown above. Devices shall implement either this interface or an IVI published interface that provides the same functionality to satisfy the method and property requirements of the IVI Core Driver Specification.

> **Observation**: > The intent of permitting drivers to implement other IVI published interfaces is to permit drivers that are migrated from earlier IVI .NET architectures to retain their implementations of this functionality.

.NET-specific Notes (see *IVI Driver Core Specification* for general requirements):

- The return value for `ErrorQuery` is null if there is no error.
- The return value for `GetSupportedInstrumentModels` is an array of strings that returns the list of supported models, one per array element.
- Drivers are permitted to implement a Set accessor on `Simulate`. However, if they

do so, they shall properly manage the driver state when turning simulation on and off.

Direct IO Properties and Methods

Per the *IVI Driver Core specification*, IVI Drivers for instruments that have an ASCII command set such as SCPI shall also provide API for sending messages to and from the instrument over the ASCII command channel. This section specifies those properties and methods.

IVI does not publish an interface for Direct IO. Drivers may implement these wherever in the driver hierarchy it makes sense.

Required Driver API (IVI Driver Core)	Core IVI.NET API
I/O Timeout	Property: Timeout (read/write)
Read Bytes	Method: ReadBytes
Read String	Method: ReadString
Write Bytes	Method: WriteBytes
Write String	Method: WriteString

.NET Property Prototypes

```
Timespan Timeout { get; set; }
<type-See Note> Session { get; }      // Optional
Byte[]ReadBytes();
String ReadString();
void WriteBytes(Byte[] buffer);
void WriteString(String data);
```

Notes:

- The *optional* `Session` property should return a strongly typed session for the underlying IO library.

Assembly Level Attributes

.NET Assembly Attributes shall be included for:

- `AssemblyTitle`. `AssemblyTitle` shall be the file name of the assembly.
- `AssemblyDescription`. `AssemblyDescription` shall be free-form; it is required to exist, but it does not have a particular value.

- `AssemblyCompany`. `AssemblyCompany` shall be the driver vendor's company name.
- `AssemblyCulture`. `AssemblyCulture` shall be "" for assemblies that are not globalized. If localized, `AssemblyCulture` shall be the value of the culture to which the assembly is localized.
- `AssemblyProduct`. `AssemblyProduct` shall be "IVI <*DriverIdentifier*> .NET Assembly".
- `AssemblyVersion`. `AssemblyVersion` shall be the same as the version resource values for the same item, as defined in *File Versioning* in *IVI Driver Core Specification*.
- `AssemblyFileVersion`. `AssemblyFileVersion` shall be the same as the version resource values for the same item, as defined in *File Versioning* in *IVI Driver Core Specification*.

.NET Assembly Attributes should be included (if applicable) for:

- `AssemblyCopyright`
- `AssemblyTrademark`

  Observation: > In MS Build projects the attributes may be generated automatically by MS Build, so will be in the MS Build file and not source code.

Packaging Requirements for .NET 6+

IVI.NET drivers are provided as NuGet packages provided via a NuGet feed. Vendors may operate their own feed or use public ones such as https://www.nuget.org.

Generally, drivers should follow the guidance provided by Microsoft for NuGet package properties. However, IVI requires:

- Versions be specified per *IVI Driver Core Specification*, File Versioning section, however:
  - No field for "Internal Version" may be included
  - Consistent with the Microsoft guidance, *"-Suffix"* may be appended to indicate pre-release versions.
- Driver NuGet packages shall specify the license terms. Drivers may use any of the common open-source licenses or use explicit licenses.

Drivers that use explicit licenses shall include the text of the license in the NuGet package and reference it in the NuGet metadata license element.

Drivers may be dual-targeted. Driver NuGet packages for dual-targeted .NET Core drivers include .NET Framework reference assemblies to enable end-customers to develop dual-targeted applications.

> **Observation:** > Drivers, and their NuGet packages, may have dependencies on tools that are not included in the NuGet package dependencies. For instance, a driver may require a license manager due to its use of licensed software.

Signing

IVI Driver NuGet packages shall be signed. The driver vendor shall publish the public key. The same key should be used to sign all the IVI.NET drivers provided by a driver vendor.

Driver suppliers shall also sign their drivers.

Contents

All IVI.NET driver NuGet packages shall include the following files:

- The assembly containing the driver
- An XML IntelliSense file
- Readme File (`README.md`)

The name of the assembly for the main driver executable shall be <*Namespace*> followed by ".dll". For example, if the driver namespace is `Agilent.Ag34401A`, the name of the dynamic link library shall be `Agilent.Ag34401A.dll`.

IVI Driver NuGet Package Fields

The IVI.NET driver NuGet package should follow Microsoft recommendations regarding NuGet package Fields.

IVI Driver General NuGet Package Tags

The IVI.NET driver NuGet package SHALL have the following NuGet tags:

Tag indicating IVI conformance: IVI-Driver-NET-Conformant

Tag indicating IVI registration: Registered-IVI-Driver

Tag indicating component type: IVI-Driver (or IVI-Driver-SharedComponents)

IVI.NET Driver Conformance

IVI.NET Drivers are required to conform to all of the rules in this document. They are also required to be registered on the IVI website.

Drivers that satisfy these requirements are IVI.NET drivers and may be referred to as such.

Registered conformant drivers are permitted to use the IVI Conformant Logo.

Driver Registration

Driver providers wishing to obtain and use the IVI Conformance logo shall submit to the IVI Foundation the driver compliance document described in *IVI Driver Core Specification*, Section Driver Conformance along with driver information and a point of contact for the driver. The information shall be submitted to the IVI Foundation website. Complete upload instructions are available on the site. Driver vendors who submit compliance documents may use the IVI Conformant logo graphics.

The IVI Foundation may make some driver information available to the public for the purpose of promoting IVI drivers. All information is maintained in accordance with the IVI Privacy Policy, which is available on the IVI Foundation website.