IVI Driver Core Specification

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

Abstract

This is the *IVI Driver Core Specification*. It describes requirements common to all IVI Core Drivers, regardless of implementation langage. IVI provides additional specifications that detail the requirements of drivers for specific languages.

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.

Warranty

Trademarks

Table of Contents

Overview of the IVI Driver Core Specification

This is the *IVI Driver Core Specification*. It describes requirements common to all IVI Core Drivers, regardless of implementation langage. IVI provides additional specifications that detail the requirements of drivers for specific programming languages.

Several other documents and specifications are related to this specification. Other documents including language-specific specifications and other extensions are used with this core standard. See the IVI web site at www.ivifoundation.org for a complete list of IVI standards.

These specifications are intended for developers of drivers and tools. Many companies build products and systems around these specifications. This document specifies the requirements for instrument drivers and areas that are open for driver designers to optimize the driver for their particular instrument.

Specification Conventions

The section describes editorial conventions used in this document and related IVI specifications.

Substitutions

This specifications use 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. Sometimes the meaning is self-evident, and no further explanation is given. The following list includes those that may need additional explanation:

- `<DriverIdentifier>`: This string uniquely identifies the driver. For example, "Agilent34410".
- `<CompanyName>`: The name of the driver vendor (not the instrument manufacturer). For example, "Agilent Technologies, Inc".
- `<RcName>`: The name of a repeated capability. Repeated capabilities may be defined in class specs or by driver developers.

Where it is important to indicate the case of substituted text, casing is indicated by the case of the text between the brackets.

- `<FooBar>` indicates Pascal casing. For example, "IviDmm".
- `<fooBar>` indicates camel casing. For example, "iviDmm".
- `<foobar>` indicates all lower case. For example, "ividmm".
- `<FOOBAR>` indicates all upper case. For example, "IVIDMM".
- `<FOO_BAR>` indicates upper snake case (all upper case with underscores between words). For example, "IVI_DMM".

Methods and Properties Nomenclature

This document uses the terms *method* and *property* to refer to elements of the API exported by an IVI driver. Unless specified otherwise, *method* refers generically to C functions and object-oriented methods. Similarly, *property* refers to C attributes and object-oriented properties.

General Requirements of IVI Core Drivers

This section specifies general requirements for IVI Core drivers.

Availability and Installation

The technology used to acquire IVI Core drivers aligns with the conventional approach used by the language. For instance, C drivers are provided with an installer and .NET 6+ drivers are provide with a NuGet package.

Language-specific specifications include the details of how drivers are acquired for a language and/or OS. The deployed driver shall include:

- Driver binaries
- Ancillary files required by development environments to use the driver binaries (for example .h files)
- Documentation, or links to documentation
- Source code with instructions for rebuilding (as required below)
- A README.md file that includes initial getting started information
- Any necessary system registrations such as those required by .NET Framework

Operating Systems and Bitness

IVI Driver language-specific specifications indicate operating systems and operating system bitness that drivers are required to support.

Drivers shall document the capabilities of the driver in the compliance document.

Source Code Availability

IVI drivers shall include source code if the source code is a simple translation of the driver calls to a separate publicly documented and officially supported interface. Drivers are exempted from this requirement if the source code includes proprietary content.

IVI drivers that include source code shall provide instructions on rebuilding the driver in at least one publicly available development environment.

> **Observation:**
>
> > Having access to source code allows a user to debug and fix instrument drivers in time critical situations when the user cannot wait for the driver supplier to fix the problem.

Repeated Capabilities

Many instruments contain multiple instances of the same type of functionality. For example, many instruments have multiple channels with independent settings. The general IVI term for functionality that is duplicated in

an instrument is *repeated capability*.

Repeated capabilities can be complex. An instrument may have multiple sets of repeated capabilities, such as channels and traces, or analog channels and digital channels. Also, repeated capabilities may be nested within other repeated capabilities, for example traces within displays. Furthermore, when working with repeated capabilities that have a large number of instances, such as digital channels, the user may find it convenient to specify sets of multiple instances of the repeated capability when calling an IVI driver API (in which case the API acts on each instance).

The IVI driver language-specific specifications specify how IVI driver APIs provide access to repeated capabilities, including nested repeated capabilities and sets of multiple repeated capability instances.

Generally, IVI Core drivers refer to repeated capabilities in one of three ways. The specific approach may be called out in a language-specific specification, or left up to the driver designer. The standard IVI API approaches for repeated capabilities are:

- **Collection Style:** Repeated capabilities can be organized into collections. Repeated capabilities can then be selected and manipulated via the collection. Languages that support collections should use collection style repeated capabilities wherever possible. This approach does not work for some languages such as C, which do not have the data structures needed to represent them.

  When using collections to represent nested repeated capabilities, each level in the hierarchy is modeled as a separate collection. To select an item in the collection, the user identifies the instance of the repeated capability for that level only. Each collection in a hierarchy is accessed separately.

- **Selector Style:** For a given repeated capability, class and specific APIs may provide a property or a method with a single parameter which selects the currently active repeated capability instance(s). All subsequent repeated capability specific operations are applied across the active repeated capability instance(s). The selected repeated capability instance(s) remains active until another instance(s) is selected.

- **Parameter Style:** For a given repeated capability, class and specific APIs may add a parameter to every method or property which uses that repeated capability. The parameter is referred to as the repeated capability selector, that "selects" the repeated capability instance(s) to be used.

Repeated capability selector types are not specified so that driver designers and IVI language-specific specifications are free to use whatever types makes the most sense in the target language. For instance, integers may make sense for a single repeated capability instance, and in Python lists may be chosen to select multiple repeated capability instances.

Simulation

Simulation is a required feature of IVI Core drivers. The *IVI Core Driver Specification* requires that simulation may be enabled or disabled at initialization.

If simulation is enabled, an IVI driver does not perform instrument I/O, and the driver creates simulated data for output parameters. Methods that return output data contain code that generates simulated data.

> **Observation:**

> > Typically, drivers generate very simple output in simulation. For example, a driver for a DMM might create simulated output data for the Read operation by generating a random number.

Multithread Safety

IVI drivers shall be multithread safe. That is, all IVI drivers shall prevent simultaneous access to a session in multiple threads of the same process from interfering with the correct behavior of the driver.

> **Observation:**
>
> > Drivers may add lock and unlock methods to facilitate usage from multiple threads.

Extent of Instrument Functionality Covered by IVI Drivers

IVI drivers for instruments that have an ASCII command set such as SCPI shall implement the full functionality of the instrument available via commands and queries with a few exceptions. Some commands and queries are not suitable for an instrument driver or could even break driver or instrument functionality if exposed to the user. For example, an IVI driver for a SCPI-based instrument might not implement SCPI commands from the following nodes:

- `DIAGnostic`
- `FORMat` (may be used internally but not exposed to users)
- `SYSTem:COMMunicate`
- Service or Factory Calibration functionality
- Undocumented SCPI (factory use only)
- Other features not normally accessed through the programmatic interface, for example:
  - `DISPlay`
  - `HARDCopy`
  - `MEMory:STATe`
  - `CURSor`

> **Observation:**
>
> > IVI driver users can send any commands to a message-based instrument using the driver's Direct I/O functions.

Direct I/O Requirement

IVI Drivers for instruments that have a supported ASCII command set, such as SCPI, shall also provide an API for sending messages to and from the instrument over the ASCII command channel. The details of the API are specified in Direct IO API.

Query Instrument Status on API Call

If the instrument can be queried for its status it shall provide a mechanism to automatically query the instrument status on every API call. This mechanism is controlled with Query Instrument Status Enabled.

Required Driver APIs

This section gives a functional description of the members of the API required for an IVI Core driver.

The general function of each member of the API is described here. However, this specification does not generally indicate what form a particular member should take.

For example:

- Initialization may be implemented as a constructor in .NET, but might be implemented as a method in another language.
- Supported Instrument Models is implemented as a method called `GetSupportedInstrumentModels()` in .NET, but might be implemented as a property in another language.

Refer to language-specific specifications for implementation details.

Initialization (Construction)

Depending on the language, IVI Core Drivers may be initialized in a constructor or in a method. This process is called *initialization* in this section.

The following parameters are defined for the *initialization* operation:

- ***Resource Name*:** An instrument specific string that identifies the address of the instrument, such as a VISA resource descriptor string.
- ***ID Query*:** Specifies if the identity of the instrument should be verified during *initialization*.
- ***Reset*:** Specifies if the instrument should be reset during *initialization*.
- ***Simulation*:** Drivers shall support a way to specify if the driver is placed in simulation mode during initialization. By default Simulate shall be false.

> **Observation:**
>
> > Drivers may implement multiple initializations that include different subsets of the parameters described here, or other parameters defined by the driver.

Driver Version

Provides a driver version string. This is a version optionally followed by a descriptive string.

The format of the version shall follow the rules for FileVersion defined in File Versioning followed by an optional string. If the string is present, a space shall separate the version from the string. The string contains additional driver specific version information. Multi-byte characters are not allowed in the string that this property returns. String characters shall be in the range of `\x20` - `\x7E`.

Examples of allowed version strings are shown below:

```
4.00.1

02.0001.12345.1 This version adds XYZ capability to the component
```

Driver Vendor

Returns the name of the vendor that supplies the IVI Core driver.

Error Query

The Error Query member queries the instrument and returns instrument specific error information.

For instruments that implement an error queue, Error Query extracts a single error from the queue.

For instruments that have status registers but no error queue, the IVI driver returns an error consistent with instrument design.

The operation of Error Query is not affected by the value of Query Instrument Status Enabled.

Instrument Manufacturer

Returns the name of the manufacturer of the instrument. The IVI driver returns the value it queries from the instrument or a string indicating that it cannot query the instrument identity. For instance, "Cannot query from instrument".

Instrument Model

Returns the model number or name of the instrument. The IVI driver returns the value it queries from the instrument or a string indicating that it cannot query the instrument identity. For instance, "Cannot query from instrument".

Query Instrument Status Enabled

IVI drivers shall implement Query Instrument Status Enabled if possible. At *initialization* Query Instrument Status Enabled shall be false, unless *initialization* options have been used to enable it.

If the instrument can be queried for its status and Query Instrument Status Enabled is True, then the driver checks the instrument status at the end of every call by the user to a method that accesses the instrument and reports an error if the instrument has detected an error. If False, the driver does not query the instrument status at the end of each user operation.

If the instrument status cannot be meaningfully queried after an operation, then this property has no effect on driver operation.

> **Observation:**
>
> > Querying the instrument status is very useful for debugging. After validating the program, the user can set this property to False to disable status checking and maximize performance.

Reset

Reset performs the following actions:

- Places the instrument in a known state. For instance, an IEEE 488.2 instrument driver may send the command string `*RST` to the instrument.
- Configures instrument options on which the IVI driver depends. A driver might enable or disable headers or enable binary mode for waveform transfers.

The user can either call Reset or specify that a reset is performed during initialization.

Simulate Enabled

Drivers shall implement a Simulate Enabled API for clients to use to read if simulation is currently active.

For details on simulation, see Simulation.

> **Observation:**
>
> > Drivers may also support changing *Simulate Enabled*, however the management of the instrument may be very complex when simulation is turned off after various driver operations have occurred.

Supported Instrument Models

Returns a list of names of instrument models with which the IVI specific driver is compatible.

The list should represent the model in exactly the same way that the IEEE 488.2 and/or LXI Identity queries return it. This includes abbreviation of the manufacturer and marketing names for the instrument.

The way that the information is returned is language-specific. For example, .NET returns the information in an array of strings.

Direct I/O

IVI Drivers for instrument that have a supported ASCII command set, such as SCPI, shall also provide an API for sending messages to and from the instrument over the ASCII command channel.

The full Direct I/O implementation shall consist of the required methods in this section plus any optional methods/properties chosen by the device. Or by providing the underlying I/O Session/Object used by the driver to perform I/O with the underlying device.

I/O Timeout

I/O Timeout is a property to read or write the I/O Timeout used for the Direct I/O operations.

Read Bytes

The Read Bytes Method reads a complete response from the instrument into an array of bytes.

Read String

The Read String method reads a complete response from the instrument and returns it as a string.

Write Bytes

The Write Bytes method writes an array of bytes to the device.

Write String

The Write String method writes a string to the device.

Documentation Requirements

The following sections detail documentation that shall be provided with IVI Core Drivers. The documentation shall include:

- Driver Introduction Documentation
- Driver Help Documentation
- Compliance Documentation
- `README.md`

Driver Introduction Documentation

Driver introduction documentation shall be a separate file named `___Introduction to <DriverIdentifier>___.<DocumentExtension>` and shall be provided with the driver. Driver introduction documentation may consist mostly of links to other documentation and may contain additional information beyond what is specified in this section.

Example Driver Introduction Document is an example driver documentation file. No trademark citations are included in this example.

Driver introduction documentation shall contain the following sections, each containing the specified information and using the section names as shown.

- **Driver Documentation:** There shall be a section called Driver Documentation. It shall list the location, file names, and purpose of the various documentation files supplied with the driver, including the compliance document (if separate).
- **Driver Source Code and Examples:** For drivers that supply source code, there shall be a section called *Driver Source Code and Examples*. It shall describe how to access the source code and examples. The section shall include a reference to the location in the documentation that explains how to build the source.

  For drivers that do not supply source code there shall be a section called *Examples*. It shall describe how to access the examples for the driver.
- **Connecting to the Instrument:** There shall be a section called Connecting to the Instrument. It shall include (or reference) documentation explaining how to get started with the driver (including opening a driver session) in different development environments.
- **Configuring Instrument Settings:** There shall be a section called Configuring Instrument Settings. It shall include (or reference) documentation of methods and properties which the user can call to to

configure instrument settings, using both properties and high-level configuration functions.

- **Configuring Driver Settings:** There shall be a section called Configuring Driver Settings. It shall include (or reference) documentation of methods and properties which the user can call to control driver operation.
- **Direct I/O:** For drivers that provide Direct I/O functions, there shall be a section called Direct I/O. It shall include (or reference) documentation of the methods and properties which the user can call to communicate directly with the instrument.
- **Instrument Command Coverage:** For drivers for message-based instruments, there shall be a section called Instrument Command Coverage. It shall include (or reference) documentation that lists the instrument commands that the driver implements, and describes the instrument commands the driver does not implement.
- **Known Issues:** For drivers with known issues, there shall be a section called Known Issues. It shall include (or reference) a list of the bugs and limitations that were known at the time the driver was released.
- **Contacting Support:** There shall be a section called Contacting Support. It shall specify how users can contact the driver vendor to report problems or ask questions relating to the driver.
- **Trademarks:** There may be a section called Trademarks. If present, it shall document any trademarks that are related to the driver or driver documentation.

Driver suppliers are encouraged to include links within each section that provide more detailed information.

The set of development environments that the driver vendor references in the document is at the discretion of the driver vendor.

Help Documentation

Help documentation shall be available with the driver. Help documentation may contain additional information beyond what is specified in this section.

For each method an IVI driver shall provide help documentation for the following:

- The method prototype
- A description of the method usage
- For each parameter, a description of its usage and valid values
- Return value and status codes
- For instruments that have an ASCII command set such as SCPI, the commands used in the method

For each property, an IVI driver shall provide help documentation for the following:

- A description of the property usage
- The data type

- Read/write access
- Valid values
- For languages that utilize return values for error and status, documentation of those values are required. For languages that utilize exceptions, driver-specific exceptions shall be documented.
- For instruments that have a supported ASCII command set such as SCPI, the commands used to get or set the property.

Common status codes or exceptions for methods and properties may be presented in a standard location instead of documented for each method and property.

Each IVI driver shall provide help documentation on known ADE restrictions, such as minimum versions or feature requirements of ADEs.

> **Observation:**
>
> > This requirement may be satisfied by the help provided by the development system used to compile the driver. For instance, Doxygen generated help based on driver source code comments may satisfy these requirements.

Documentation of Implemented SCPI Commands

IVI drivers for instruments that have a supported ASCII command set such as SCPI shall document the implemented ASCII commands and characterize the commands that are not implemented.

Partially implemented ASCII commands may be documented as "implemented".

Copyright Notice

Each IVI driver shall include the following text in a visible location in the help documentation.

> Content from the IVI specifications reproduced with permission from the IVI Foundation.
>
> The IVI Foundation and its member companies make no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The IVI Foundation and its member companies shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Compliance Documentation

Each IVI driver shall include documentation indicating the IVI specifications with which the driver complies. The compliance information shall be distributed with the driver. Compliance documentation may contain additional information beyond what is specified in this section. If 32-bit and 64-bit versions of the same driver exist, the contents of the compliance documents for them shall be the same, and shall describe both.

In the compliance document, operating systems shall be designated by the accepted name (e.g. Windows 10, Windows 11) followed by version information in parentheses. Version information may be service packs, versions, or builds, as appropriate. If the OS and version is available in multiple bitnesses, the bitness precedes

the OS name. For example, 32-bit Windows 10, version 21H2 would be listed as "32-bit Windows 10 (21H2)", where "21H2" is the applicable version. Windows 11, which is 64-bit only, would be listed as "Windows 11 (24H1)".

> **Observation:**
>
> > The compliance document has several items that refer to driver bitness. This reflects the bitness of processes in which the driver may be executed. For some driver implementation languages such as .NET, the driver executable may not be bitness-specific. However, if a driver uses libraries that are only available in a specific bitness, the driver will be limited to executing in processes of that bitness. For example, if a driver uses a support library that is only available as a 64-bit library, the driver will only be able to execute in a 64-bit process.

The compliance documentation shall contain the following sections, each containing the specified information and using the section and item names as shown.

Example Compliance Document contains an example of a compliance document.

Compliance Category Section

The Compliance Category section of the compliance document informs the user of the IVI specifications with which it complies. All IVI drivers shall include a list of those specifications in this section.

This list shall include the *IVI Driver Core Specification* and any language-specific specification with which it complies, as well as other relevant specifications. If driver functionality is version-specific, the specification version should be included.

Driver Identification Section

The Driver Identification section informs the user of the identity of the driver. All IVI drivers shall include the following items in this section.

- **Driver Version:** The Driver Version item shall contain the value of the Driver Version that the IVI driver returns.
- **Driver Vendor:** The Driver Vendor item shall contain the value of the Driver Vendor that the IVI driver returns.
- **Driver Bitness:** The Driver Bitness item shall list all driver bitnesses supported by the driver. Valid values are 32-bit and 64-bit.
- **Source Code:** The Source Code item shall specify if instrument driver source code is available to end-users and the conditions under which it is distributed. When source code is not required, (for instance, if the instrument driver is just a layer on top of support libraries), this item shall indicate why the source is not available.

Optionally, drivers may include other information as well, such as the driver's driver identifier, or a description.

Hardware Information Section

The Hardware Information section informs the user about the hardware supported by the instrument driver. All IVI drivers shall include the following items in this section:

- **Instrument Manufacturer:** The Instrument Manufacturer item shall contain the name(s) of the instrument manufacturer.
- **Supported Instrument Models:** The Supported Instrument Models item shall contain the value of the Supported Instrument Models property that the IVI driver returns.
- **Supported Bus Interfaces:** The Supported Bus Interfaces item shall contain an itemized list of the bus interfaces that the IVI driver supports.

Software Prerequisites Section

The Software Prerequisites section informs the user about additional software required by the instrument driver.

In rare cases, software prerequisites may vary according to the driver bitness In such cases, an item's title may be followed by "for 32-bit drivers" or "for 64-bit drivers". For example, if the supported operating systems are different for 32-bit and 64-bit, this section would include the items *Supported Operating Systems for 32-bit Drivers* and *Supported Operating Systems for 64-bit Drivers*.

For each software information section, IVI drivers shall include the following:

- **Supported Operating Systems:** The Supported Operating Systems item shall contain a list of all operating systems on which the IVI driver is supported at the time of release.
- **Driver Dependencies:** The Driver Dependencies item shall contain a list of the support libraries that the IVI driver requires but that are not provided by the IVI Foundation or the operating system. Necessary restrictions, such as the minimum version number, should be included.

Unit Testing Section

The Testing section informs the user about the unit testing performed on the instrument driver. All IVI drivers shall include the following items in this section:

Test Setup

The compliance document shall contain one or more Test Setup sections. Each Test Setup section shall specify one or more test setups on which you ran the complete set of unit tests, as specified in Section Unit Test Procedure.

If you performed the complete set of unit tests on multiple setups, you may be able to express the multiple setups in one Test Setup section by listing multiple values for the various setup items, as long as you test all valid combinations implied by the all the multiple values listed.

- **Instrument Model (Firmware Version):** The Instrument Model and Firmware Version item shall specify the instrument models and their firmware versions on which you performed the complete set of unit tests.

- **Bus Interface:** The Bus Interface item shall specify the bus interface through which the instrument was connected to the computer when you performed the complete set of unit tests.
- **Operating System:** The Operating System item shall specify the operating system(s) on which you performed the complete set of unit tests.
- **Driver Bitness:** The Bitness Bitness item shall specify the bitness of the application which you used to perform the complete set of unit tests.
- **VISA Vendor and Version:** The VISA Vendor and Version item shall specify the vendor and version of the VISA implementation used to perform the complete set of unit tests if the driver requires VISA. If the driver does not use VISA, this item may be omitted.
- **IVI Shared Components Version** If this driver uses IVI Shared Components, the IVI Shared Components Version item shall fully specify the type of the shared components (IVI, IVI.NET, or IVI.NET Core) and the version that was used to perform the complete set of unit tests.

> **Observation:**
>
> > To clarify the implication of listing multiple values for multiple items, consider the following partial example:
> >
> > Operating System (Version): 64-bit Windows 10 (21H1), Windows 11 (23H2)
> >
> > Driver Bitness: 32-bit, 64-bit
> >
> > This indicates that you performed the complete set of unit tests using the following combinations:
> >
> > - 64-bit Windows 10 with a 32-bit application
> > - 64-bit Windows 10 with a 64-bit application
> > - Windows 11, with a 32-bit application
> > - Windows 11, with a 64-bit application

The Example Compliance Document contains two Test Setup items. The first documents testing with one instrument over a GPIB for a broad range of OS and driver bitnesses. The second documents testing with another instrument over USB and LAN for one OS (Windows 11) and one driver bitness (64-bit).

Driver Installation Testing Section

The Driver Installation Testing section informs the user about the driver installation testing performed on the instrument driver. All IVI drivers shall include the following items in this section:

- **Operating System:** The Operating System item shall specify the operating system(s) on which you performed the driver installation testing.

Driver Buildability Section

The Driver Buildability section informs the user about the driver buildability testing performed on the instrument driver. All IVI drivers that include source code shall include the following items in this section:

- **Operating System:** The Operating System item shall specify the operating system(s) on which you performed the driver buildability testing.

Driver Test Failures Section

The Driver Test Failures section informs the user about the failures that the driver testing revealed on the instrument driver that were not fixed before release. All IVI drivers shall include the following items in this section:

- **Known Issues:** The Known Issues item shall specify a list of known issues that reflect all test failures that were not fixed. If there are no known issues, indicate None.

Additional Compliance Information Section

The Additional Compliance Information section informs the user about additional information that relates to compliance but is not included in other defined items. This section is not present if no additional compliance information exists.

Alternate Compliance Documentation Formats

Drivers that otherwise comply with the *IVI Driver Core Specification* may use the format described in *IVI-3.1 Architecture Specification*. That format shall be an acceptable alternate format for drivers that conform to that specification.

README.md Documentation

The driver README.md file includes details that will help users get started with the driver. It shall contain:

- The location of the `___Introduction to <DriverIdentifier>___.<DocumentExtension>` documentation file.
- The location of the help documentation file(s).
- The text of the compliance documentation. The compliance documentation includes items normally associated with a `README.md` file, such as driver version information, known issues, and driver dependencies. The `README.md` file may also contain the location of a separate compliance document that also contains the compliance documentation.

The `README.md` file may contain additional information that users may need to know, or find useful, before using the driver.

File Versioning

IVI driver Windows DLLs shall contain a Windows version resource with the three entries: `CompanyName`, `ProductName`, and `FileVersion`.

- `CompanyName` is a string that contains the value of the Driver Vendor or Driver Vendor property that the IVI driver returns.

- `ProductName` is a string that includes the value of the Driver Identifier property that the IVI driver returns.

- `FileVersion` is a string in the following format:

  `MajorVersion.MinorVersion.BuildVersion[.InternalVersion]`

`MajorVersion`, `MinorVersion`, `BuildVersion`, and `InternalVersion` are decimal numbers greater than or equal to zero and less than or equal to 65535. The MajorVersion shall not be zero, except for a pre-release version of the initial release of a driver. Each number may contain leading zeros but may not exceed 5 digits. The number fields are separated by periods, with no embedded spaces. The maximum valid FileVersion is 65535.65535.65535.65535.

A FileVersion string shall contain at least a `MajorVersion`, `MinorVersion`, and `BuildVersion`. A driver shall be released with an incremented `MajorVersion` or `MinorVersion` number when any of the following conditions are true:

- The new version of the driver contains changes in its API syntax (either breaking or non-breaking).
- The new version of the driver contains significant changes to its semantics (that is, behavior).

A driver shall be released with at least an incremented `BuildVersion` number when any of the following conditions are true:

- A new bitness of the driver is provided.
- The driver is modified in a way that does not require a `MajorVersion` or `MinorVersion` update.

If a driver vendor provides both a 32-bit and a 64-bit version of a driver for a given language and operating system, the 32-bit and 64-bit DLLs shall have the same `MajorVersion`, `MinorVersion`, and `BuildVersion`. Notice that when a 64-bit version of the driver is initially made available, the 32-bit version must be updated at the same time.

For the purpose of determining the version of an IVI driver, only the `MajorVersion`, `MinorVersion`, and `BuildVersion` are used; the InternalVersion is not used. The InternalVersion is optional and reserved for use by driver suppliers.

When comparing two `FileVersion` values, integer comparisons are performed successively for each number, starting at the leftmost number. If the first two numbers are equal, the next two numbers are compared, and so on.

Examples

The following are examples of valid `FileVersion` values:

```
1.0.0
2.00.00
01.02.03
11.22.33
5.0.1
3.14.159
4.0.1000.0
0001.1.1.00005
```

The following are examples of comparisons of `FileVersion` values for the purpose of comparing driver versions:

```
5.1.345.213 is greater than 4.3553.3244.234

5.1.345.213 is less than 5.1.346.213

5.2.13.26 is equal to 5.2.13.56

5.001.100.001 is equal to 5.1.00100.1

2.15.32.1 is greater than 2.15.18.1

1.1.1 is equal to 1.1.1.0

1.1.2 is less than 1.1.10
```

Driver Conformance

In order to claim conformance to this spec:

- Drivers shall conform to all of the rules in this document.
- Drivers shall verify conformance by following the rules in this section.
- Drivers shall conform to an IVI language-specific IVI Driver specification. Language-specific IVI specifications include IVI Generation 2014 specifications.

> **Observation:**
>
> > This specification contains a subset of the requirements in IVI Generation 2014, therefore drivers that comply with those specifications also comply with this specification.

> **Observation:**
>
> > Developers that want to support a language that does not have an IVI language-specific IVI Driver specification are encouraged to submit draft language-specific specifications to the IVI Foundation.

Driver Verification

IVI Drivers shall be evaluated and tested to verify that they meet all applicable IVI requirements. Conformance verification is necessary to verify that the driver complies with the requirements for IVI Drivers.

IVI Drivers shall include a *Compliance Document* that documents the completion of the driver verification specified in this section. See Compliance Document for details of that document.

The following sections describe the minimum testing that IVI driver suppliers shall perform on an IVI driver before releasing it and the testing documentation requirements.

Unit Test Procedure

Every entry point in the driver shall be tested in simulation mode and connected to at least one of the driver's supported instruments with simulation off. The complete unit test shall be run at least once on one *test setup* as defined below:

- Instrument Model and Firmware Version
- Bus Interface
- Operating System and Version
- OS Bitness and Application Bitness
- VISA Vendor and Version (if VISA is required by the driver)

Using the driver in simulation mode, the driver tester shall do the following:

- Call all implemented functions/methods at least once and verify that they return without failure.
- Use at least one legal value for each instrument setting and verify that the driver accepts the value(s).

Using the driver with an instrument, the driver tester shall run one or more client programs that call the driver and verify that the driver and instrument respond as expected, either by reading values back from the instrument or through external means. The client programs(s) shall test the driver in the following ways:

- The client program(s) shall test each function/method with the values listed below for each parameter that represents an instrument setting. The program(s) or tester shall verify that all output values are reasonable for each set of input parameters tested.
- At least one legal value
- Verify value limits by testing:
    - legal values at the limits of each range
    - illegal values at the limits of each range
    - one legal value within each range
- Verify parameter coercions by testing:
    - at least one value between each discrete setting
    - legal values at the limits of the entire range
    - at least one illegal value
- For parameters with a discrete set of explicitly specified legal values (such as enumerations and Booleans), each legal value and at least one illegal value
- The client program(s) shall test each property as if it were a method, that is, as the single parameter to a "Set" call and a return value from a "Get" call.

- The client program(s) shall test at least one method and one property with each repeated capability instance specifier legal for the setup and at least one illegal repeated capability instance specifier.

Driver Installation Testing

The installation shall be tested by:

- Installing the driver
- Instantiating and running the driver
- Testing every example included with the driver

Installation testing shall be done on at least two operating systems, unless the driver only supports a single operating system. Unless the driver supports only one bitness, at least one tested operating system shall be 32-bit, and at least one shall be 64-bit. Each operating system tested must have been updated with a version that is within 6 months of being the most recent.

Driver Buildability Testing

For drivers that include source code, the driver tester shall re-build the driver from the installed source code according to the documented instructions. The driver tester shall verify that the re-built driver works on at least one combination of operating system version and bitness.