



IVI-3.4: API Style Guide

April , 2002 Edition
Revision 1.0

Important Information

The API Style Guide (IVI-3.4) is authored by the IVI Foundation member companies. For a vendor membership roster list, please visit the IVI Foundation web site at www.ivifoundation.org, or contact the IVI Foundation at 2515 Camino del Rio South, Suite 340, San Diego, CA 92108.

The IVI Foundation wants to receive your comments on this specification. You can contact the Foundation through email at ivilistserver@ivifoundation.org, through the web site at www.ivifoundation.org, or you can write to the IVI Foundation, 2515 Camino del Rio South, Suite 340, San Diego, CA 92108.

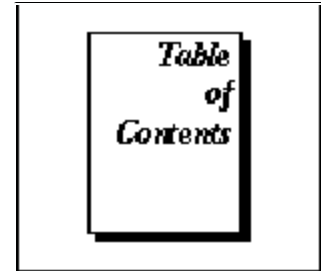
Warranty

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.

Trademarks

Product and company names listed are trademarks or trade names of their respective companies.

No investigation has been made of common-law trademark rights in any work.



1.	Overview of the API Style Guide.....	7
1.1	Introduction.....	7
1.2	Overview.....	7
1.3	References	7
1.4	Definitions of Terms and Acronyms	7
2.	Approach to Designing Instrument Class Interfaces.....	8
2.1	Development Process	8
2.2	Scope of an Instrument Class Specification.....	8
2.3	Attributes	8
2.3.1	Coupled	8
2.3.2	Uncoupled.....	9
2.4	Functions	9
2.4.1	Configuration	9
2.4.2	Action.....	9
2.4.3	Retrieve Measurement Data.....	9
2.5	Relationship of Attributes to Configuration Functions.....	9
3.	Naming Conventions.....	10
3.1	Instrument Class Names	10
3.2	Capability Group Names	10
3.3	Function Names	10
3.4	Attribute Names	11
3.5	Parameter Names	11
3.6	IVI-C.....	11
3.6.1	Functions	11
3.6.2	Attributes	11
3.6.3	Defined Values	12
3.7	IVI-COM.....	12
3.7.1	Interface.....	12
3.7.2	Methods	12
3.7.3	Properties	12
3.7.4	Defined Values	12
3.7.5	Interface Reference Properties	13
4.	Parameter Types.....	14
4.1	Integers.....	14
4.2	Reals.....	14
4.2.1	Continuous Ranges and Discrete Values.....	14
4.2.2	Infinity and Not A Number.....	15

4.3	Enumerations	15
4.3.1	IVI-C.....	15
4.3.2	IVI-COM.....	15
4.4	Strings.....	16
4.4.1	IVI-C.....	16
4.4.2	IVI-COM.....	16
4.5	Booleans.....	16
4.6	Arrays.....	16
4.6.1	IVI-C.....	16
4.6.2	IVI-COM.....	17
4.7	Pointers.....	17
4.8	Sessions.....	17
4.8.1	IVI-C.....	18
4.8.2	IVI-COM.....	18
5.	Version Control.....	19
5.1	IVI-COM Interface Versioning.....	19
5.2	IVI-C Interface Versioning.....	19
6.	COM IDL Style	20
6.1	Use of retval.....	20
6.2	Use of out vs. in,out.....	20
6.3	Use of SAFEARRAY as a Property.....	20
6.4	Help Strings.....	20
7.	Controlling Automatic Parameter Setting	21
8.	Time Parameters	22
9.	Units	23
10.	Disable	24
11.	Completion Codes and Error Messages	25
11.1	IVI-C.....	25
11.2	IVI-COM.....	25
12.	Repeated Capabilities	26
12.1	Required Attributes	26
12.1.1	Item.....	26
12.1.2	Count	26
12.1.3	Name (COM only).....	27
12.2	Required Functions.....	27
12.2.1	Get Name (C only).....	27

13.	Hierarchies.....	28
13.1	C Function Hierarchy.....	28
13.1.1	Sample Function Hierarchy	29
13.2	C Attribute Hierarchy	30
13.2.1	Sample Attribute Hierarchy.....	30
13.3	COM Interface Hierarchy	32
14.	Synchronization	34
14.1	Non-blocking.....	34
14.2	Blocking.....	34
15.	Instrument Class Specification Layout	35
15.1	Overview Layout.....	35
15.2	Capabilities Groups Layout.....	36
15.3	General Requirements Layout.....	36
15.4	Capability Group Section Layout.....	36
15.4.1	Attribute Section Layout.....	37
15.4.2	Function Section Layout	39
15.5	Attribute ID Definitions Layout.....	40
15.6	Attribute Value Definitions Layout.....	41
15.7	Function Parameter Value Definitions Layout.....	41
15.8	Error and Completion Code Value Definitions Layout	42
15.9	Hierarchies	43
15.9.1	COM Hierarchy	43
15.9.2	C Function Hierarchy.....	45
15.9.3	C Attribute Hierarchy	45
15.10	Appendix A: IVI Specific Driver Development Guidelines Layout	47
15.11	Appendix B: Interchangeability Checking Rules Layout.....	47
15.12	Appendix C: ANSI C Include File	47
15.13	Appendix D: COM IDL File.....	47
16.	Expressing Tone	48
16.1	Requirement	48
16.2	Recommendation.....	48
16.3	Permission	49
16.4	Possibility and Capability.....	49

API Style Guide

API Style Guide Revision History

This section is an overview of the revision history of the Ivi-3.4 specification.

Table 1-1. Ivi-3.4 Revisions

Revision Number	Date of Revision	Revision Notes
Revision 0.1	December 17, 1999	Original draft.
Revision 0.2	January 31, 2000	Noticed a few things. Added various things related to COM.
Revision 0.3	February, 2000	Based on discussion at the February meeting.
Revision 0.4	September, 2000	Based on discussion at the May and August meeting.
Revision 0.5	November, 2001	Version taken to November meeting.
Revision 0.6	February, 2001	Based on discussion at the November 2000 meeting.
Revision 0.7	April, 2001	Based on discussion at the February 2001 meeting.
Revision 0.8	May 24, 2001	
Revision 1.0vc1	August 17, 2001	Final draft, voting candidate 1.
Revision 1.0vc2	October 2001	Final draft, voting candidate 2.
Revision 1.0	December 2001	Voting candidate 5 review draft. Changes based on discussion at December IVI meeting.

1. Overview of the API Style Guide

1.1 Introduction

This specification is primarily written to direct the efforts of working groups writing the *IVI-3.2 Inherent Capabilities Specification*, *IVI-3.3 Standard Cross-Class Capability Specification*, shared components, and instrument class specifications. Specifications should have a common format, similar presentation of content, and consistent design of capability groups. By providing a consistent style, IVI specific driver writers will find information presented in a familiar manner and thus avoid overlooking requirements. By adhering to this style, specification writers should avoid accidentally omitting information.

IVI specific drivers will invariably contain functions and attributes not described in the instrument class specification. Driver writers are expected to follow the style described here for those functions and attributes. Throughout the style guide names are shown that contain a class name. For IVI specific drivers, the same rules apply except the class name is replaced by the instrument specific prefix.

1.2 Overview

This specification applies to all IVI Instrument Class Specifications approved after this specification is approved. It provides rules and recommendations on organization, naming, choosing parameter types, required guidance to driver writers, and more.

- ? Existing instrument class specifications also provide insight into appropriate style. The following instrument class specifications, however, were adopted before this specification was completed:
 - IviScope
 - ? IviDmm
 - ? IviFgen
 - ? IviDCPwr
 - ? IviSwtch

Therefore these specifications may use a style different from the one in this specification.

1.3 References

Several other documents and specifications are related to this specification. These other related documents are as follows:

- ? *VXIplug&play* VPP-3.2 Instrument Driver Functional Body Description
- ? *VXIplug&play* VPP-3.4 Instrument Driver Programmatic Developer Interface Specification
- ? IVI-3.2: IVI Inherent Capabilities Specification
- ? IVI-3.3: Standard Cross-Class Capabilities Specification
- ? IEEE 488.2-1987 IEEE Standard Codes, Formats, Protocols, and Common Commands.

1.4 Definitions of Terms and Acronyms

This specification does not define additional terms or acronyms. Refer to *IVI-5: Glossary* for a description of terms used in this specification.

2. Approach to Designing Instrument Class Interfaces

Instrument Class working groups face many decisions while writing a specification. Some of these issues have been faced and dealt with by previous working groups and their wisdom can be applied to future work.

2.1 *Development Process*

Every specification goes through a process from its inception to approval to maintenance. This process is defined and controlled by the technical working group. Contact its chairman for details.

2.2 *Scope of an Instrument Class Specification*

One of the first questions a working group should answer is "What features are appropriate to include in this instrument class?" This answer defines the instrument class.

From this set of features, a few comprise the base capability. These features are commonly found in all instruments of this instrument class.

The remaining features are grouped into extension groups. Creating appropriate extension groups and allocating features to them is a difficult process which requires the insight of many experts who understand a variety of instruments in the instrument class. Placing too many features in one extension group may prevent a particular instrument driver from implementing that group. All the instruments in a class may provide a large variety of triggering methods though no one instrument implements all the methods. Each triggering method thus belongs in its own extension group. If one feature, however, fundamentally requires another feature then both belong in the same extension group.

Features found in only a small number of instrument models should not be included in the instrument class. No gain in interchangeability is achieved if only one or two instruments have a particular extension group. The user should access these specialized features through driver-specific means.

During early instrument class development, features may be added or removed. They may migrate between capability groups. As the specification matures, movement of features should decrease and finally cease.

2.3 *Attributes*

An instrument's state can generally be modeled with a vector of attributes. The instrument class specification defines a set of attributes within each capability group.

Each attribute can be set separately. Groups of attributes can also be set in a single configure function call.

Attributes are classified in two types depending on whether its legal range depends on other attributes' values.

2.3.1 *Coupled*

Coupled attributes affect each other. The legal settings for an attribute coupled to another attribute depend on the other attribute's setting. Instrument classes provide a configuration function which sets all the attributes which are coupled to each other. Thus the user can set all the coupled attributes at once and the driver and instrument can sort out the coupling.

For example, the function and range in a digital multimeter are coupled. While 1e6 is likely a valid range when the function is resistance it is not likely valid when the function is DC volts. So the IviDmm instrument class provides a function to set both attributes in one call.

2.3.2 Uncoupled

The legal settings of an uncoupled attribute do not depend upon any other instrument settings. Uncoupled attributes may appear in configuration functions.

For example, the trigger slope in a digital multimeter is independent of any other attribute.

2.4 **Functions**

The functions described by the API fall into two categories.

2.4.1 Configuration

A Configuration function sets one or more attributes. These functions are especially useful when dealing with coupled attributes. The driver resolves couplings to ensure that when the user specifies a legal state that the instrument reaches that state without error. Sometimes this resolution is done with the assistance of processing capabilities in the instrument itself. Other times the driver resolves any coupling conflicts itself before programming the instrument.

2.4.2 Action

An Action routine makes the instrument perform an operation which depends on the present configuration. The operation generally does not involve changing any attribute's value. Triggering a measurement or starting a sweep are examples of action functions.

2.4.3 Retrieve Measurement Data

Two forms of a function which retrieve measurement data are commonly found in IVI drivers:

Fetch - The instrument returns already available measurement data. If the instrument is acquiring data, the driver waits until the acquisition is complete before returning data.

Read - The instrument initiates a new measurement and waits for the measurement to complete before returning data.

2.5 **Relationship of Attributes to Configuration Functions**

For IVI-C, the instrument class specification should define for each attribute exactly one high level configuration function. A possible exception is an attribute which can be changed in several capability groups. Often, several configuration functions control single, uncoupled attributes. While a user could use the Set Attribute function, these configuration functions provide better visibility into the functionality of the instrument class and instrument.

For IVI-COM, an uncoupled attribute may not have a corresponding high level configuration function. The attribute can be configured by setting the corresponding COM property.

3. Naming Conventions

Naming things and spelling those names can be a surprisingly contentious subject. As the English language has demonstrated, inconsistent spelling leads to frustration. The rules and recommendations here are intended to reduce a user's frustration level when discovering how the large variety of names used in IVI drivers are determined.

Avoid abbreviations. If a name contains so many words that it becomes objectionably long, typically more than 50 characters, abbreviations are allowed. If a word is abbreviated in one name, abbreviate that word everywhere and abbreviate it the same way in all names. Always abbreviate maximum as max and minimum as min.

Use acronyms only when they are more understandable than the complete word. Capitalize acronyms as if they are single words, e.g. CW, RTD, RADAR.

3.1 Instrument Class Names

An instrument class name may contain up to thirty-one total characters. The first three characters shall be "Ivi" using mixed case. The remaining characters are left to the discretion of the instrument class specification working group. The only constraint is that the resulting instrument class name cannot conflict with any existing instrument class name.

The size limit results from the constraints on the Class Driver Prefix attribute described in Section 5.5, *Class Driver Prefix* in *IVI-3.2: Inherent Capabilities Specification*. The string that this attribute returns contains a maximum of 32 bytes including the NUL byte.

IMPORTANT: Since this name is prolific throughout a instrument class specification, it should be as short as possible while leaving no ambiguity about the nature of the instruments in the class.

Two special forms of the instrument class name are used throughout this document where the instrument class name is substituted into a name.

<ClassName> means substitute the instrument class name with mixed case. For example, IviDcPwr.

<CLASS_NAME> means substitute the instrument class name in all upper-case characters. For example, IVIDCPWR.

3.2 Capability Group Names

A capability group name may contain an arbitrary number of characters. The first characters shall be the instrument class name, <ClassName>. The capability group is described using complete English words. These words are appended to the instrument class name with no spaces between the words and the first character of each word is capitalized.

For example, IviDcPwrSoftwareTrigger. The prefix, IviDcPwr, is the instrument class name. The complete English words, Software and Trigger, describe the capability group.

3.3 Function Names

A function name consists of one or more complete English words which describe the operation performed. Refer to existing instrument class specifications for commonly used naming patterns. The first word should be a verb or at least imply performing an action.

For example, Configure Edge Trigger Source.

A reference to a function shall use the one or more complete words, with the first character of each word capitalized and the words separated by a space. Use a Times font for generic references to functions. Do not use the IVI-C or IVI-COM form of the name.

3.4 Attribute Names

Construct attribute names by using complete English words which describe what is controlled. Be concise, but include enough words to avoid ambiguity.

Be consistent with word order. If a word appears in multiple attributes, put it first before any modifying words. Words which describe the core functionality come first, followed by words which qualify the previous word. A useful side effect of these rules becomes apparent when attributes are listed alphabetically. Related attribute name and values are listed together. For example in Waveform Size Min, waveform is the core functionality. Size qualifies waveform and Min qualifies size.

Random order frustrates. Having one attribute named Max Waveform Size and another Waveform Size Min would be extremely annoying.

For example, TV Trigger Signal Format.

A reference to an attribute shall use the one or more complete words, with the first character of each word capitalized and the words separated by a space. Use a Times font for references to attributes. Do not use the IVI-C or IVI-COM form of the name.

3.5 Parameter Names

Do not use the instrument class name in parameter names. Capitalize each word in a parameter's name regardless of the number of words in the name.

Use multiple words to make a parameter's usage obvious. Parameter names in a function prototype are a form of documentation.

Use the same parameter name in both COM method prototypes and C prototypes for the same function.

For example, InputImpedance.

3.6 IVI-C

This section refines the rules in Sections 3.3-5 for IVI-C.

3.6.1 Functions

IVI function names in the C prototype shall consist of <ClassName>, an underscore character, and the function name with the spaces removed. The only underscore character is the one after the instrument class name.

For example, the function, Configure Edge Trigger Source becomes
IviScope_ConfigureEdgeTriggerSource.

3.6.2 Attributes

Since attributes names are macros they use only upper case characters. The full attribute name begins with:
<CLASS_NAME>_ATTR_

The remainder of the IVI-C attribute name is the attribute name in all upper-case characters with spaces replaced by underscores.

For example, TV Trigger Signal Format becomes IVISCOPE_VAL_TV_TRIGGER_SIGNAL_FORMAT

3.6.3 Defined Values

Some properties and function parameters have defined values. These defined value names are macros, so they use only upper-case characters. A defined value name begins with:

```
<CLASS_NAME>_VAL_
```

The remainder of the name is also in all upper case. If the remainder uses multiple words, they are separated by underscore characters.

For example, `IVISCOPE_VAL_NTSC`.

3.7 **IVI-COM**

This section refines the rules in Sections 3.3-5 for IVI-COM.

3.7.1 Interface

The names for all interfaces defined by IVI start with `IIVI`.

For interfaces defined in an IVI instrument class, the interface names begin with `I<ClassName>`. The root interface name has nothing more added to the instrument class name. For other interfaces, additional words are added which, when possible, are the same as one of the levels in the C function hierarchy.

For example, `IIVI_Scope` is the root interface of the `IviScope` instrument class and `IIVI_ScopeReferenceLevel` is one of the interfaces defined in the `IviScope` instrument class.

3.7.2 Methods

The hierarchy context plus the method name shall communicate the same content as the generic function name. The method name may omit some of the words in the function name if the interface pointer names provide the same information.

The COM method prototype is of the form:

```
HRESULT <MethodName> ( various parameters );
```

For example, the function, `Configure Edge Trigger Source` becomes `Trigger.Edge.Configure`.

3.7.3 Properties

The hierarchy context plus the property name shall communicate the same content as the attribute name. The property name may omit some of the words in the attribute name if the interface pointer names provide the same information.

IVI-COM Property Names do not have any prefix. They are generally the same as the attribute name with the spaces removed except when some of the words are omitted.

For example, `Class Driver Major Revision` becomes `ClassDriverMajorRevision` and `TV Trigger Signal Format` becomes `Trigger.TV.SignalFormat`

Do not use the interface name or its corresponding interface property name in the property name. For example a property in `IIVI_DmmTrigger` would be `Count` rather than `TriggerCount`.

3.7.4 Defined Values

Defined value names are contained within a COM enumeration. An IVI-COM enumeration name is of the form:

```
<ClassName><descriptive words>Enum
```

Each word is capitalized with no spaces or underscores. The suffix, Enum, shall be included.

For example, `IviScopeTriggerCouplingEnum`

The defined values within the enumeration are of the form:

```
<ClassName><words from enumeration name><words>
```

The same words from the enumeration name shall appear in every value for a particular enumeration. The trailing word or words shall be picked to differentiate the values.

For example, `IviScopeTriggerCouplingAC` and `IviScopeTriggerCouplingDC`.

3.7.5 Interface Reference Properties

A special property is one which points to an interface. The names of these properties shall be the trailing words of the type of interface the property points to. The instrument class name prefix, `I<ClassName>` is omitted.

For example, a property which points to a interface of type `IviScopeReferenceLevel` is named `ReferenceLevel`.

4. Parameter Types

Functions and methods pass parameters which always have a type. Attributes and properties have a type. Using the right type in the right situation helps a user intuitively know a parameter's type.

The type of a parameter or an attribute may be given using a general term or a language specific type. The other types can be inferred using Table 4-1. *Compatible Data Types for IVI Drivers* in *IVI-3.1: Driver Architecture Specification*. Instrument class specifications often use just the ANSI C type.

4.1 Integers

Things which can be naturally counted should be integers. If a parameter can be thought of as a number of things, use an integer. The number of points in a trace or the number of triggers are naturally integers.

	input	output
C	ViInt32	ViInt32*
COM	LONG	LONG*

In some cases, instruments transfer large arrays, several million elements, and the individual data items fit into a 16-bit integer. Under this unusual circumstance, the instrument class specification writers may specify the data as a ViInt16 for C and SHORT for COM.

4.2 Reals

Any continuous value should be a real. If something can logically ever be a non-integer value, make it a real. Anything with physical units, such as voltage, power, frequency, length, angle, etc. should be a real.

	input	output
C	ViReal64	ViReal64*
COM	DOUBLE	DOUBLE*

In some cases, instruments transfer large arrays, several million elements, and the individual data items fit into a 32-bit floating point number. Under this unusual circumstance, the instrument class specification writers may specify the data as a ViReal32 for C and FLOAT for COM.

4.2.1 Continuous Ranges and Discrete Values

Instrument classes should specify discrete legal values for a parameter only when intermediate values are not meaningful. In such cases, the instrument class shall specify that coercion is not allowed. IVI specific drivers report an error when a user specifies a value not in the set of legal values. Instrument classes should encourage IVI specific drivers to do the same where the IVI class compliant specific driver takes a subset of the values defined by the instrument class.

In general, instrument classes should allow continuous ranges of values for real-valued parameters and attributes, even if some instruments in the instrument class implement only a discrete set of values for the setting. Some instruments that implement only a discrete set of values might accept a continuous range of values and coerce user-specified values to the discrete set. Other instruments might accept only the discrete set, in which case the IVI specific driver accepts a continuous range and coerces user-specified values to the discrete set accepted by the instrument. The instrument class should specify whether the value from the user should be coerced up, down, to the nearest arithmetic value, or to the nearest geometric value. If an instrument performs coercion in a manner different from what the instrument class specifies, the IVI specific driver coerces the value before sending it to the instrument.

Instrument classes should not specify a minimum or maximum value for a continuous range unless the value reflects an inherent limitation of the setting. An example of a setting with an inherent limitation is the vertical range on an oscilloscope, which cannot be a negative number.

4.2.2 Infinity and Not A Number

Use the IEEE 754 representations for positive infinity (PInf), negative infinity (NInf) and not a number (NaN). Instrument classes shall specify the exact circumstances when these values are returned. For example, a function could be required to return NaN when a measurement value could not be generated.

A shared component, defined in *IVI-3.12: Floating Point Services Specification*, is available for driver writers to get these defined values. IVI drivers use the shared component to reduce the possibility of incompatibility with other implementations.

For functions that can return one of these special values, the instrument class shall provide functions which compare any number against these special values. Don't require the user to make comparisons with NInf or PInf or Nan, but provide functions which do the comparison inside the driver.

4.3 Enumerations

An enumeration is a good choice for a parameter which has a natural association to several discrete possibilities and the possibilities have no numeric content. If adding or removing a possibility is difficult, the parameter should probably not be an enumeration, but rather an integer or real. For example, HIGH, MEDIUM, and LOW are poor choices as removing High would leave Medium and Low. Likewise, adding a choice creates a badly named value such as Medium High. In this case, the parameter should be a real or integer with appropriate coercion to a value the instrument can handle.

4.3.1 IVI-C

The names of the values associated with an enumeration shall have a prefix of the form:

<CLASS_NAME>_VAL_

The remainder of the enumeration value name should be easily associated with the parameter or attribute being set.

	input	output
C	ViInt32	ViInt32*

4.3.2 IVI-COM

Enumerations in IVI-COM are defined using a typedef in the IDL. Every typedef for an enumeration shall include these attributes:

public - so the alias becomes part of the type library

v1_enum - so the enumerated type is a 32-bit entry, rather than the 16-bit default.

The IDL for enumerations is of the form:

```
typedef [public,
        v1_enum,
        HELP( "ENUM_<enum_name>" ) ]
enum prefix<enum_name>Enum
{
    prefix<enum_name>Choice1      = 0,
    prefix<enum_name>Choice2      = 1,
```

```
} prefix<enum_name>Enum;
```

Where <enum_name> is COM enumeration name generated using the rules in Section 3.7.4, *Defined Values*.

4.4 Strings

Handling strings is generally cumbersome and they should be avoided especially for input parameters. Users get spelling and word order wrong very easily. Parsing an input string is generally harder than handling a numeric type and interchangeable parsers may be too much to expect. If only a finite choice of strings make sense, use an enumeration.

Input strings are necessary when something has a name, such as a channel or trace.

Output strings whose only use is to be read by the user are, however, a good choice. Strings elucidating an error code are quite useful. The string is generated by the driver or forwarded from the instrument and the user is never expected to programmatically parse it, only display it where a human can parse it.

	Input	Output
C	ViConstString	ViChar[]
COM	BSTR	BSTR *

4.4.1 IVI-C

In IVI-C, include a BufferSize parameter before the string parameter. If <string> is the name of the string parameter, the BufferSize parameter shall be ViInt32 <string>BufferSize. Since the string is null terminated, an actual size parameter is not required.

The IVI driver can expect the user to allocate at least as many characters for the string as the value of BufferSize. If the value of the string parameter is VI_NULL, the value of BufferSize may be zero.

Instrument class specifications shall also follow the rules in Section 3.1.21 *Additional Compliance Rules for C Functions with ViChar Array Output Parameters* in IVI 3.2: *Inherent Capabilities Specification*.

4.4.2 IVI-COM

The IVI driver allocates the memory for a BSTR so a size parameter is not needed.

4.5 Booleans

Any parameter which might otherwise be an enumeration but could only ever have two values is most likely a Boolean. Use the associations:

0 (VI_FALSE and VI_OFF) for False, Off, and No

1 (VI_TRUE and VI_ON) for True, On, and Yes

	input	Output
C	ViBoolean	ViBoolean*
COM	VARIANT_BOOLEAN	VARIANT_BOOLEAN*

4.6 Arrays

Passing arrays has the problem of specifying the size of the array both on input and output.

4.6.1 IVI-C

When passing an array into a function, each array shall have two parameters: a ViInt32 which is the size of the array and a pointer of appropriate type to the array. The size parameter immediately precedes the pointer in the parameter list.

When retrieving an array from a function, each array shall have three parameters: a `ViInt32` which is the size of the array allocated by the caller, a pointer of appropriate type to the array, and finally a pointer to a `ViInt32` which the function fills with the actual number of values returned.

Sometimes multiple arrays of exactly the same size are passed into or out of a function. In this case, pointers to these additional arrays shall immediately follow the first array pointer without any additional size parameters.

If the size of the array is the same for all applications, the size parameter may be omitted. The instrument class specification shall specify the size and provide a macro with its value.

Remember, the calling program is responsible for allocating any memory.

Use `Vi<type> <name>[]` for array parameters. The name of the parameter used to pass in the size of the array is `ViInt32 <name>ArraySize`. The name of the parameter used to pass back the actual number of elements put into the array is `ViInt32* <name>ActualSize`.

The function prototype for a routine which has an array as an input would have the form:

```
ViStatus <ClassName>_ConfigureArray(ViSession vi,
                                   ViInt32  AbcArraySize,
                                   ViReal64  Abc[]);
```

The function prototype for a routine which has one array as an output would have the form:

```
ViStatus <ClassName>_FetchArray(ViSession vi,
                                ViInt32  AbcArraySize,
                                ViReal64  Abc[],
                                ViInt32*  AbcActualSize);
```

The function prototype for a routine which has two arrays of the same size as outputs would have the form:

```
ViStatus <ClassName>_FetchArrays(ViSession vi,
                                 ViInt32  AbcArraySize,
                                 ViReal64  AbcDEF[],
                                 ViReal64  AbcXYZ[],
                                 ViInt32*  AbcActualSize);
```

4.6.2 IVI-COM

COM interfaces shall pass arrays using `SAFEARRAY`. The size is embedded in the structure so no additional parameters are needed.

4.7 Pointers

Do not pass pointers as values. Addresses are used only when passing

- ? a parameter by reference,
- ? an array,
- ? a string,
- ? or a COM interface pointer.

4.8 Sessions

A valid session is created when the Initialize function in an IVI driver is called.

4.8.1 IVI-C

If one of the parameters of a function in an IVI-C driver is a session it shall be the first parameter and its type shall be ViSession.

4.8.2 IVI-COM

Generally, COM methods do not have an explicit session parameter. The session is maintained within a particular object. Class references serve a similar purpose as sessions.

5. Version Control

The IVI Foundation may publish revised versions of the instrument class specifications. As IVI drivers are written, the IVI Foundation may discover areas to improve the specifications including adding additional capability groups. These new versions shall be done in a manner which does not make existing applications inoperable.

5.1 *IVI-COM Interface Versioning*

The IVI Foundation shall create a new version of any standard IVI-COM interface that changes in any of the following ways:

- ? The interface name is changed.
- ? The interface inheritance is changed.
- ? A method or property is added or deleted, or the spelling of a method or property name is changed.
- ? The parameter list of a method or property is changed, or the spelling or data type of any parameter is changed.
- ? Any IDL attribute of any element of the interface is changed.
- ? For enumerations used in the interface,
 - o An enumeration is added or deleted, or the spelling of an enumeration value name or tag is changed.
 - o An enumeration value is added or deleted, or the integer value associated with an enumeration value name is changed.
 - o Any IDL attribute of any element of an enumeration is changed.

In general, the only change that does not require the creation of a new version of an interface is the modification of an IDL help string.

Each interface shall have a base name that is version independent. The first version shall use this name without modification. For each subsequent version, the base name shall have an integer appended, starting with “2” and incrementing by 1. For example, the first version of the IviScope’s trigger interface is named IviScopeTrigger. The second published version of this interface would be named IviScopeTrigger2, the third would be named IviScopeTrigger3, and so on.

Whenever a new version of an interface is created, it shall be assigned a new IID.

5.2 *IVI-C Interface Versioning*

The IVI Foundation shall maintain backwards compatibility when modifying the IVI-C interface defined in a instrument class specification. The IVI Foundation may extend an IVI-C interface, but not modify the existing elements. Modifications that shall be avoided include:

- ? Changes to existing function prototypes
- ? For an existing attribute,
 - o Changes to its data type
 - o Changes to its association with a repeated capability
 - o Removing read or write access
- ? Changes to existing attribute ID constant names and values
- ? Changes to existing value ID constant names and values
- ? Changes to existing error code constant names and values

The IVI-C interface may be extended by adding new functions, attributes, attribute values, read or write attribute access, and error codes.

6. COM IDL Style

One of the responsibilities of a working group developing an instrument class specification is to create the COM IDL for the instrument class. An IVI-COM specific driver writer also produces COM IDL. These style considerations apply to both.

6.1 *Use of retval*

Many methods have a single out parameter. This parameter should also be declared as the *retval* for the function. That is, the attributes for the parameter are `[out, retval]`. Declaring a *retval* parameter allows the user to set a variable to the value of that parameter using an assignment operator. Declaring a parameter to be a *retval* also avoids the problem related in Section 6.2.

If a method has multiple out parameters, typically none of the parameters is declared as the *retval*.

6.2 *Use of out vs. in,out*

Microsoft Visual Basic has a defect which causes a memory leak if a parameter is declared to be only `[out]`. It does not free the memory allocated by the server. To work around this defect, no parameters in IVI IDL shall every be declared as just `[out]`. They shall always be either `[out, retval]` or `[in, out]`.

If a parameter is declared as `[in, out]` when it truly is just an output, the instrument class specification shall state so.

This defect is known to exist up through version 6.0. If the defect is repaired in future revisions of Visual Basic, this requirement may be amended.

6.3 *Use of SAFEARRAY as a Property*

Properties shall never be a `SAFEARRAY` of any type since Microsoft Visual Basic does not gracefully handle `SAFEARRAYS` as properties.

6.4 *Help Strings*

The IDL file for an instrument class specification shall contain the following help string.

```
"<ClassName> <Revision> Type Library"
```

Where `<Revision>` is the revision of the instrument class specification.

This help string appears prominently in various tools that browse for available type libraries. The IDL file shall contain help context IDs and help strings for every interface, method, property, and enumeration.

7. Controlling Automatic Parameter Setting

Instruments and drivers contain algorithms which automatically adjust settings based on other settings or characteristics of input signals. Typically, these algorithms can be enabled and disabled. For example, a DMM can adjust its voltage range based on the amplitude of the input signal. The setting which enables or disables the algorithm is separate from the setting which represents the actual value. For example, even with autoranging on, the DMM range has an actual value. Another example is automatic coupling of frequency span to resolution bandwidth, video bandwidth, and sweep time in a spectrum analyzer.

An instrument class specification shall provide a separate parameter and associated attribute which controls automatic coupling algorithms. Do not use special, out-of-range values for the attributes which are automatically controlled. For example, do not use a range value of -1 to turn on autoranging. Instead have two attributes: Autorange Policy and Range.

The values for the Autorange Policy should be an enumeration not a Boolean. Some possible values include on, off, once, and manual. The instrument class specification writers determine which values are appropriate.

8. Time Parameters

All time parameters, except time out parameters shall be real and have units of seconds. This rule is consistent with Section 4.2, *Reals*. Time out parameters, however, shall be an integer with units of milliseconds.

When an instrument class specification defines a value for timeout parameter, its C name shall be of the form:

<CLASS_NAME>_VAL_MAX_TIME_<WORD>

and the COM name shall be of the form:

<ClassName>MaxTime<Word>

Two common values are Immediate and Infinite. Place a table of this form in the function section.

<i>Name</i>	<i>Description</i>	
	<i>Language</i>	<i>Identifier</i>
Max Time Immediate	The function returns immediately. If the operation had not already completed, the function returns an error.	
	C	<CLASS_NAME>_VAL_MAX_TIME_IMMEDIATE
	COM	<ClassName>MaxTimeImmediate
Max Time Infinite	The function waits indefinitely for the operation to complete.	
	C	<CLASS_NAME>_VAL_MAX_TIME_INIFINITE
	COM	<ClassName>MaxTimeInfinite

Place a table of this form in the Function Parameter Value Definitions section.

<i>Value Name</i>	<i>Language</i>	<i>Identifier</i>	<i>Actual Value</i>
Max Time Immediate	C	<CLASS_NAME>_VAL_MAX_TIME_IMMEDIATE	0
	COM	<ClassName>MaxTimeImmediate	0
Max Time Infinite	C	<CLASS_NAME>_VAL_MAX_TIME_INIFINITE	0xFFFFFFFFFUL
	COM	<ClassName>MaxTimeInfinite	0xFFFFFFFFFUL

A timeout parameter name shall be MaxTimeMilliseconds to convey to the user that its units are milliseconds.

9. Units

Most likely every parameter or attribute which is real, and even some integers, has a unit associated with it. The instrument class specification shall specify what that unit is. Instrument classes may provide mechanisms to change the unit of a value.

Avoid units with a multiplier. Stick with the base unit. The unit meter is preferred over kilometer.

The unit should, in general, be an SI primary unit. IEEE 488.2 Table 7-1 *<suffix unit> Elements* provides a list of commonly used units.

10. Disable

The instrument class specification shall include a statement about how IVI specific drivers implement the Disable function for that instrument class. The statement shall appear in Section 3.1.1. See Section 6.4, *Disable* in *IVI-3.2: Inherent Capabilities Specification* for a general description of the function.

11. Completion Codes and Error Messages

Section 9, *Common Error and Completion Codes* in *IVI-3.2: Inherent Capabilities Specification* describes completion codes and error messages which are applicable to most of the error cases in IVI drivers and other components. They cannot, however, cover all cases. Instrument class specification writers shall examine the codes in *IVI-3.2* for appropriate codes. When those codes are not sufficient, the instrument class may define additional codes and messages.

Along with the code the instrument class specification shall define an error message which the IVI driver returns in Error Message, in C or in ErrorInfo in COM.

11.1 *IVI-C*

The C constant name for an error shall begin with:

`<CLASS_NAME>_ERROR_`

The C constant name for a warning shall begin with:

`<CLASS_NAME>_WARN_`

The remainder of the name shall be a series of words, all in uppercase, separated by underscore characters. This portion of the name shall exactly match the equivalent portion of the COM constant name.

11.2 *IVI-COM*

The COM constant name for an error shall begin with:

`E_<CLASS_NAME>_`

The COM constant name for a warning shall begin with:

`S_<CLASS_NAME>_`

The remainder of the name shall be a series of words, all in uppercase, separated by underscore characters. This portion of the name shall exactly match the equivalent portion of the C constant name.

12. Repeated Capabilities

Many instruments have capabilities which are duplicated. For example, an oscilloscope might have several channels with identical functionality. A power supply may have several outputs. Several techniques are available to provide consistent access to the repeated capabilities.

1. Add a parameter to every function which uses a repeated capability. This parameter is generally a string
2. Provide a function which selects a currently active capability. It remains active until another one is selected. The function uses a string parameter as the selector.
3. In COM, collections provides a natural way of differentiating between capabilities.

In IVI-C, only the first two choices are viable. In IVI-COM, collections are generally a good choice when option 1 is used in IVI-C and only one repeated capability is controlled at a time.

If option 2, the selector, is used in the IVI-C API also use it IVI-COM API. Do not mix the selector technique with either the parameter or collection technique.

Do not use the selector technique when the API contains nested repeated capabilities. A nested repeated capability occurs when the user must specify at least two items in a hierarchy to gain access to a particular capability. For example, an API with a window repeated capability where each window has a trace repeated capability contains a nested repeated capability.

See Section 3, *Repeated Capability Group* in *IVI 3.3: Standard Cross-Class Capabilities Specification*, for a complete description of the attributes and functions.

12.1 Required Attributes

For every repeated capability, the instrument class specification shall define several attributes.

12.1.1 Item

This attribute is present in IVI-COM only when the repeated capability is implemented as a collection. This attribute is not present in IVI-C.

The actual name of the attribute is the name of the repeated capability followed by the word Item. For example, in an instrument class where channels are a repeated capability the name would be Channel Item.

The attribute requires a string parameter which is the name of one of the repeated capabilities. It returns an interface pointer which can be used to control the attributes and other functionality of that particular repeated capability.

12.1.2 Count

This attribute is present in both IVI-C and IVI-COM for all three techniques.

The actual name of the attribute is the name of the repeated capability followed by the word Count. For example, in an instrument class where channels are a repeated capability the name would be Channel Count.

The attribute contains how many of the capability exist in the IVI specific driver.

12.1.3 Name (COM only)

This attribute is present in IVI-COM for all three techniques.

In COM, the name of the method is Name. Which capability the method is associated with is determined by its placement in the hierarchy.

The read-only property requires an integer parameter which is the index into the repeated capabilities. It returns a string which is the name of a repeated capability. The value of the index must be greater than or equal to one and less than or equal to Count. Zero is not a legal value.

12.2 **Required Functions**

For every repeated capability, the instrument class specification shall define at least one function.

12.2.1 Get Name (C only)

This function is present in IVI-C for all three techniques. It shall not appear in IVI-COM

In C, the actual name of the functions is the word Get followed by the name of the repeated capability followed by the word Name. For example, in an instrument class where channels are a repeated capability the name of the function would be GetChannelName.

The function requires an integer parameter which is the index into the repeated capabilities. It returns a string which is the name of a repeated capability. The value of the index must be greater than or equal to one and less than or equal to Count. Zero is not a legal value.

13. Hierarchies

Instrument class specifications and the resulting IVI drivers present many functions and attributes to the user. In an effort to reduce the perceived complexity, these items are organized into hierarchies. Creating hierarchies with different styles add unneeded confusion to the user. Instrument class specification writers and IVI specific driver developers should follow these guidelines when designing hierarchies.

13.1 C Function Hierarchy

IVI-C drivers include a function panel file that contains and a function hierarchy of the exported functions. A function hierarchy assists the user in finding functions by organizing the functions into categories.

The function panel file format is specified in Section 6, *Function Panel File Format*, of the *VXIplug&play* specification *VPP-3.3: Instrument Driver Interactive Developer Interface Specification*.

Instrument class specifications provide guidelines for organizing functions defined by the instrument class specification. IVI-C specific drivers should follow the hierarchy for the instrument class defined functions and extend the hierarchy to include instrument specific functions.

General principles to follow when creating a function hierarchy:

1. The Initialize, Initialize With Options, and Close functions should appear at the top level of the function tree hierarchy.
2. Functions should be organized into class types according to usage. Common categories include *Application*, *Configuration*, *Measurement*, and *Action/Status*.
 - *Application* functions are created from a lower-level set of IVI driver functions. Typically, these are provided with IVI specific drivers, but are not defined within an instrument class specification.
 - *Configuration* functions change the state of instrument settings.
 - *Action/Status* functions include two types of functions. Action functions, such as Initiate and Abort, cause the instrument to initiate or terminate test and measurement operations. These operations can include arming the triggering system or generating a stimulus. These functions are different from the configuration functions because they do not change the instrument settings, but only order the instrument to carry out an action based on its current configuration. Status functions obtain the current status of the instrument or the status of pending operations.
 - *Measurement* functions capture single-point and multi-point measurement data. Functions that initiate and transfer data to and from the instrument appear within the *Measurement* category. If a *Measurements* category exists for a class, low-level action functions should not appear under a separate *Action/Status* category. Instead, low-level action and measurement functions, such as Initiate, Fetch, Abort, and Sent Software Trigger should appear in a *Low-Level Measurements* sub-category below the *Measurements* category. If the instrument operation includes more than scalar measurements, this category might be more appropriately named. For example, the IviScope specification defines a *Waveform Acquisition* category that includes both waveform and measurement functions.
 - *Utility* functions perform operations that are auxiliary to the operation of the instrument. These utility functions include inherent functions specified by IVI 3.2: Inherent

Capabilities Specifications. Other functions that appear under the *Utility* category include functions for reading/writing to the instrument and calibration functions.

3. Within a given category, place the functions in the natural order in which they will be used. Functions that should be called first appear higher in the hierarchy than those that are called later. For example, Initialize appears before Close.
4. If a category contains many functions, divide the category into sub-categories. For example, the Configuration category could contain a sub-category called Trigger Configuration.
5. All attribute accessor functions should appear in a *Set/Get Attribute* sub-category of the *Configuration* category.

13.1.1 Sample Function Hierarchy

A sample function hierarchy appears in **Table 13-1**. IVI Class and IVI specific drivers should follow this hierarchy as closely as possible.

Table 13-1 Prefix Function Hierarchy

Name or Class	Function Name
Initialize	<ClassName>_init
Initialize with Options	<ClassName>_InitWithOptions
Configuration...	
<Configure sub-categories and functions>	
Set/Get Attribute...	
Set Attribute...	<ClassName>_SetAttribute<type>
Get Attribute...	<ClassName>_GetAttribute<type>
Measurement...	
Read <Measurement>	<ClassName>_Read<Measurement>
<Measurement sub-categories and functions>	
Low-Level Measurement...	
Initiate <Measurement>	<ClassName>_Initiate<Measurement>
<Measurement> Status	<ClassName>_<Measurement>Status
Fetch <Measurement>	<ClassName>_Fetch<Measurement>
Abort	<ClassName>_Abort
Utility...	
Reset	<ClassName>_reset
ResetWithDefaults	<ClassName>_ResetWithDefaults
Disable	<ClassName>_Disable
Self-Test	<ClassName>_self_test
Revision Query	<ClassName>_revision_query
Error-Query	<ClassName>_error_query
Error Message	<ClassName>_error_message
Get Next Coercion Record	<ClassName>_GetNextCoercionRecord

Table 13-1 Prefix Function Hierarchy

Name or Class	Function Name
Get Next Interchange Warning	<ClassName>_GetNextInterchangeWarning
Clear Interchange Warnings	<ClassName>_ClearInterchangeWarnings
Reset Interchange Check	<ClassName>_ResetInterchangeCheck
Invalidate All Attributes	<ClassName>_InvalidateAllAttributes
Error Info...	
Get Error	<ClassName>_GetError
Clear Error	<ClassName>_ClearError
Locking...	
Lock Session	<ClassName>_LockSession
Unlock Session	<ClassName>_UnlockSession
Close	<ClassName>_close

13.2 C Attribute Hierarchy

IVI-C drivers include a sub file that contains and an attribute hierarchy of the exported attributes. An attribute hierarchy assists the user in finding attributes by organizing attributes into logical groups.

The sub file format is specified in Section 7, *Function Panel Sub File Format*, of the *VXIplug&play* specification *VPP-3.3: Instrument Driver Interactive Developer Interface Specification*.

Instrument class specifications provide guidelines for organizing attributes defined by the instrument class specification. IVI-C specific drivers should follow the hierarchy for the instrument class defined functions and extend the hierarchy to include instrument specific functions.

General principles to follow when creating an attribute hierarchy:

1. The Inherent IVI Attributes category appears before instrument class-defined attribute categories.
2. Place attributes in a natural order in which they will be set. Attributes that should be set first appear higher in the hierarchy than those that are set later. For example, in the IviDmm class, Range should always be set before Resolution. Therefore, the Range attribute appears before Resolution in the hierarchy.
3. Attributes should be grouped by extension capability groups.
4. Attributes specific to a particular trigger type should be grouped together. For example, in the IviScope class, each trigger type appears as a level 2 category. Each trigger category includes the attributes unique to that trigger type.
5. Attributes that are necessary to fully specify the state of the instrument appear higher in the hierarchy than those that are optional or included in an extension capability group. For example, for the IviDmm class-specification, the Function and Range attributes are at level 2 while the Thermocouple Type attribute, which is used only when taking temperature measurements, appears at level 3.

13.2.1 Sample Attribute Hierarchy

A sample attribute hierarchy appears in **Table 13-2**. Insofar as it is practical, IVI Class and IVI specific drivers should follow this hierarchy.

Table 13-2. <ClassName> C Attributes Hierarchy

Category or Generic Attribute Name	C Defined Constant
<i>Inherent IVI Attributes</i>	
<i>User Options</i>	
Range Check	<CLASS_NAME>_ATTR_RANGE_CHECK
Query Instrument Status	<CLASS_NAME>_ATTR_QUERY_INSTRUMENT_STATUS
Cache	<CLASS_NAME>_ATTR_CACHE
Simulate	<CLASS_NAME>_ATTR_SIMULATE
Record Value Coercions	<CLASS_NAME>_ATTR_RECORD_COERCIONS
Interchange Check	<CLASS_NAME>_ATTR_INTERCHANGE_CHECK
<i>Class Driver Identification</i>	
Class Driver Description	<CLASS_NAME>_ATTR_CLASS_DRIVER_DESCRIPTION
Class Driver Prefix	<CLASS_NAME>_ATTR_CLASS_DRIVER_PREFIX
Class Driver Vendor	<CLASS_NAME>_ATTR_CLASS_DRIVER_VENDOR
Class Driver Revision	<CLASS_NAME>_ATTR_CLASS_DRIVER_REVISION
Class Driver Class Spec Major Version	<CLASS_NAME>_ATTR_CLASS_DRIVER_CLASS_SPEC_MAJOR_VERSION
Class Driver Class Spec Minor Version	<CLASS_NAME>_ATTR_CLASS_DRIVER_CLASS_SPEC_MINOR_VERSION
<i>Driver Identification</i>	
Specific Driver Description	<CLASS_NAME>_ATTR_SPECIFIC_DRIVER_DESCRIPTION
Specific Driver Prefix	<CLASS_NAME>_ATTR_SPECIFIC_DRIVER_PREFIX
Specific Driver Locator	<CLASS_NAME>_ATTR_SPECIFIC_DRIVER_LOCATOR
Specific Driver Vendor	<CLASS_NAME>_ATTR_SPECIFIC_DRIVER_VENDOR
Specific Driver Revision	<CLASS_NAME>_ATTR_SPECIFIC_DRIVER_REVISION
Specific Driver Class Spec Major Version	<CLASS_NAME>_ATTR_SPECIFIC_DRIVER_CLASS_SPEC_MAJOR_VERSION
Specific Driver Class Spec Minor Version	<CLASS_NAME>_ATTR_SPECIFIC_DRIVER_CLASS_SPEC_MINOR_VERSION
<i>Driver Capabilities</i>	
Supported Instrument Models	<CLASS_NAME>_ATTR_SUPPORTED_INSTRUMENT_MODELS
Class Group Capabilities	<CLASS_NAME>_ATTR_GROUP_CAPABILITIES
<i>Instrument Identification</i>	
Instrument Manufacturer	<CLASS_NAME>_ATTR_INSTRUMENT_MANUFACTURER
Instrument Model	<CLASS_NAME>_ATTR_INSTRUMENT_MODEL
Instrument Firmware Revision	<CLASS_NAME>_ATTR_INSTRUMENT_FIRMWARE_REVISION

Table 13-2. <ClassName> C Attributes Hierarchy

Category or Generic Attribute Name	C Defined Constant
<i>Advanced Session Information</i>	
Logical Name	<CLASS_NAME>_ATTR_LOGICAL_NAME
I/O Resource Descriptor	<CLASS_NAME>_ATTR_IO_RESOURCE_DESCRIPTOR
Driver Setup	<CLASS_NAME>_ATTR_DRIVER_SETUP
<i>Basic Operation</i>	
<Base Capability Attributes and sub-categories>	
<i>Trigger</i>	
<Trigger attributes and sub-categories>	
<Other Capability Groups/Operations>	
<Attributes and sub-categories>	

13.3 COM Interface Hierarchy

In defining some COM hierarchies, these general principles were discovered.

1. Use nouns for interface names, not verbs. These nouns may be instrument subsystem names. Extension group names may also be good choices.
2. Create large interfaces only when the items are all read-only or seldom used. A user’s level of discomfort seems to be around fifteen. Never exceed 30 as this many items requires scrolling.
3. Create interfaces with at least three items. One interface with one or two items is acceptable, but rare.
4. Separate multiple, mutually exclusive options into separate sub-interfaces. For example, with scope triggers only one of the several trigger options can be used at any one time so they were split into their own sub-interfaces.
5. Put only the items that are truly repeated in the repeated capability interface, the interface with the singular name, to minimize redundancy.
6. Avoid adding other methods and properties to the collections interfaces, the interface with the plural name. Any additional methods and properties beyond the standard three should apply to all the items in the collection.
7. Separate or combine optional elements depending on factors such as the number of methods and properties, the depth of hierarchy, etc. Examples are multi-point trigger on DMM, which is separated, and AMInternal in the Fgen, which is not separated.
8. Represent cross-class capabilities in the same way.
9. Keep in mind the potential for future evolution of the design and potential ways that specific instruments might extend the instrument class capabilities when designing the hierarchy.
10. Keep logical siblings at the same depth in the hierarchy. Either put the elements in one interface or in sibling interfaces.
11. Hide seldom used, read-only item lower in the hierarchy, particularly if it serves to highlight more commonly used properties and methods.

12. Avoid adding properties and methods to the root. Instrument class specifications should generally add only interface reference properties to the root. Some instruments classes, however, have a small number of methods and properties which are so fundamental to the operation of the instrument class that the only logical place in the hierarchy is at the root. Range, function, and resolution for the Dmm are examples.
13. Maintain parallelism between like elements with respect to the level in the hierarchy and interface names.

These principles may lead to conflicting results. Instrument class writers should use them, but with caution and judgment. These principles have no known order or priority.

14. Synchronization

Users must often synchronize a program's execution with the operation of instruments in a system. Instrument class specifications can help users by supplying methods which assist in synchronization. Using similar notation, behavior, and parameters is a matter of style and not functionality.

When instrument class specification writers identify an operation which can take an appreciable amount of time to complete, they should consider adding two functions to the capability group associated with the operation.

14.1 *Non-blocking*

This function should be of the form:

```
Is<operation complete> (ViBoolean* Done)
```

The words in <operation complete> are related to the operation being performed. For example, Settled.

The function returns rapidly. The parameter, Done, is false if the operation is still being performed and it is true if the operation has completed.

14.2 *Blocking*

This function should be of the form:

```
WaitUntil<operation complete> (ViInt32 MaxTimeMilliseconds)
```

The words in <operation complete> are identical to those in the non-blocking function.

The function does not return until the operation has completed or more than MaxTimeMilliseconds of time has elapsed. The return value indicates whether the operation completed or too much time elapsed before completion.

15. Instrument Class Specification Layout

Each instrument class specification shall contain the sections in Table 15-1.

Table 15-1 Section Labeling

Section Number	Section Topic
1	Overview
2	<ClassName> Class Capabilities
3	General Requirements
4	Base Capability Group
5	First Extension Group
...	
n	Last Extension Group
n+1	Attribute ID Definitions
n+2	Attribute Value Definitions
n+3	Function Parameter Value Definitions
n+4	Error and Completion Code Value Definitions
n+5	Hierarchies

Each instrument class specification shall contain the appendixes in Table 15-2.

Table 15-2 Appendix Labeling

Appendix Letter	Appendix Topic
A	Specific Driver Development Guidelines
B	Interchangeability Checking Guidelines
C	ANSI C Include File
D	COM IDL File

Tables and figures are numbered <main section>-<sequential number>. Numbers re-start at one in each major section.

Insert a page break before every first level section.

15.1 Overview Layout

Each Overview section shall contain the subsections:

- 1.1 Introduction
- 1.2 <ClassName> Class Overview
- 1.3 References
- 1.4 Definitions of Terms and Acronyms

Define terms which are specific to the instrument class. Terms of more general interest are defined in *IVI-5: Glossary*. When an instrument class specification working group discovers a term of general interest, it should submit the term to the glossary working group.

15.2 Capabilities Groups Layout

Each Capability Group section shall contain the subsections:

- 2.1 Introduction
- 2.2 <ClassName> Group Names
- 2.3 Repeated Capability Names

If the instrument class does not define any repeated capabilities, section 2.3 shall contain the text “The <ClassName> Class Specification does not define any repeated capabilities.” If it does define one or more repeated capabilities, they shall be listed in section 2.3 and section 2.3 shall contain a sub-section for each different repeated capability describing how the name appears in the Configuration Store.

15.3 General Requirements Layout

Each General Requirements section shall contain the subsections:

- 3.1 Minimum Class Compliance
 - 3.1.1 Disable Function
- 3.2 Capability Group Compliance

Section 3.1 shall include a statement that an IVI specific driver shall comply with the instrument class specification, IVI: 3-1, and IVI: 3.2.

If an instrument class needs to describe additional compliance rules for inherent capabilities beyond the requirements in *IVI-3.2: Inherent Capabilities Specification*, they shall appear in Sections 3.1.x. A section is added for each inherent capability with additional requirements.

15.4 Capability Group Section Layout

Each capability group, base and extensions, section shall contain the following subsections:

n.1 Overview:

An overview of the capability group that describes its purpose and general use.

n.2 Attributes: (Optional)

Defines the attributes that are a part of the capability group. For each attribute the capability group defines the name of the attribute, the data type, the access (read and write - R/W, or read only - RO), a description, defined values, and additional compliance requirements. Refer to Section 15.4.1, Attribute Section Layout for more information regarding the layout of attribute subsections. There is one attribute section for each attribute in a capability group.

If the instrument class capability group does not contain any attributes, the section is omitted.

The title of section shall be <Capability Group Name> Attributes.

After the list of attributes, include this paragraph:

This section describes the behavior and requirements of each attribute. The actual value for each attribute ID is defined in Section n+1, <ClassName> Attribute ID Definitions.

n.3 Functions: (Optional)

Defines the functions that are part of the capability group. For each function the capability group defines the function name, description, input parameters, output parameters, completion codes, and additional compliance requirements. Refer to Section 15.4.2, Function Section Layout for

more information regarding the layout of function subsections. There is one function section for each function in a capability group.

If the instrument class capability group does not contain any functions, the section is omitted.

The title of section shall be <Capability Group Name> Functions.

After the list of functions, include this paragraph:

This section describes the behavior and requirements of each function.

n.4 Behavior Model:

Defines the relationships between IVI driver attributes and functions with instrument behavior.

The title of section shall be <Capability Group Name> Behavior Model.

n.5 Group Compliance Notes: (Optional)

Section 3, General Requirements defines the general rules an IVI specific driver must follow to be compliant with a capability group. This section specifies additional compliance requirements and exceptions that apply to a particular capability group.

If the instrument class capability group does not contain any additional compliance requirements, the section is omitted.

The title of section shall be <Capability Group Name> Compliance Notes.

Adjust the section numbers as needed. If an optional section does not exist do not skip a level 2 heading number. For example, if a capability has no attributes, the Functions section is numbered n.2.

15.4.1 Attribute Section Layout

Insert a page break before each attribute section. Each Attribute section shall contain the following subsections:

Capabilities Table:

A table with five columns with the following titles:

Data Type:

Specifies the data type of the attribute. See Table 15-3. *Compatible Data Types for IVI Drivers* in *IVI-3.1: Driver Architecture Specification* for a complete list of allowed data types.

Access:

Specifies the kind of access the user has to the attribute. Possible values are RO, WO, and R/W.

? R/W (read/write) – indicates that the user can get and set the value of the attribute.

? RO (read-only) – indicates that the user can only get the value of the attribute.

? WO (write-only) – indicates that the user can only set the value of the attribute.

Applies to:

Specifies whether the attribute applies to the instrument as a whole or applies to a repeated capability. The field contains either the name of the repeated capability or N/A. Examples of repeated capability names are Trace, Display, and Channel.

Coercion:

Some attributes represent a continuous range of values, but allow the IVI specific driver to coerce the value that the user requests to a value that is more appropriate for the instrument. For these cases the specification defines the direction in which the IVI specific driver is allowed to coerce a value. Possible values are Up, Down, and None.

- ? Up – indicates that an IVI specific driver is allowed to coerce a user-requested value to the nearest value that the instrument supports that is greater than or equal to the user-requested value.
- ? Down – indicates that an IVI specific driver is allowed to coerce a user-requested value to the nearest value that the instrument supports that is less than or equal to the user-requested value.
- ? None – indicates that the IVI specific driver is not allowed to coerce a user-requested value. If the instrument cannot be set to the user-requested value, the IVI specific driver must return an error.

Any other kind of coercion is indicated with a documentation note.

High Level Function(s):

Lists all high level functions that access the attribute.

For example,

Data Type	Access	Applies to	Coercion	High Level Functions
ViReal64	R/W	N/A	None	Configure Acquisition Record

COM Property Name:

The name of the COM property as it appears in the IDL file. The name of the property is preceded by all the interface reference pointers, separated by dots, needed to reach this method in the hierarchy.

COM Enumeration Name:

If the value of the COM property is an enumeration, its name appears here. If its not an enumeration, then N/A is appears.

C Constant Name:

The constant identifier used to access the attribute and defined in the include file.

Description:

Describes the attribute and its intended use.

Defined Values: (Optional)

Defines all the attribute values that the instrument class specifies for the attribute using a table with two main columns. The left column is labeled Name and contains the generic name for a value and the right column Description. Under the Description column are entries for the Identifier used in every language of interest to IVI.

For example,

Name	Description	
	Language	Identifier
Current Trip	The power supply disables the output when the output current is equal to or greater than the value of the Current Limit attribute.	
	C	IVIDCPWR_VAL_CURRENT_TRIP

	COM	IviDcPwrCurrentLimitTrip
Current Regulate	The power supply restricts the output voltage such that the output current is not greater than the value of the Current Limit attribute.	
	C	IVIDCPWR_VAL_CURRENT_REGULATE
	COM	IviDcPwrCurrentLimitRegulate

The actual numeric value associated with a defined value is specified in the Attribute Value Definitions section.

If the attribute does not define any defined values, the section is omitted.

Compliance Notes: (Optional)

The General Requirements section defines the general rules an IVI specific driver must follow to be compliant with an attribute. This section specifies additional compliance requirements and exceptions that apply to a particular attribute.

If the attribute does not contain any additional compliance requirements, the section is omitted.

15.4.2 Function Section Layout

Insert a page break before each function section. Each Function section shall contain the following parts:

Description

Describes the behavior and intended use of the function.

COM Method Prototype:

Defines IVI-COM prototype. The name of the method is preceded by all the interface reference pointers, separated by dots, needed to reach this method in the hierarchy. As in IDL, each parameter is preceded by an indication of whether it is in, out, or retval as well as its type. For example,

```
HRESULT Measurements.IsWaveformElementInvalid ([in] DOUBLE MeasurementValue,
                                               [out, retval] VARIANT_BOOL *IsValid);
```

If a COM method does not exist, use N/A here and refer to any attribute that provides equivalent functionality.

C Prototype:

Defines the C Language prototype. For example,

```
ViStatus IviScope_IsInvalidWfmElement (ViSession Vi,
                                       ViReal64 MeasurementValue,
                                       ViBoolean *IsOverrange);
```

If a C prototype does not exist, use N/A here.

Parameters:

Describes each function parameter using one or two tables. The first table is for input parameters. It has three columns. The left column is labeled Inputs and the middle column is labeled Description. The right column is labeled Base Type.

For example:

Inputs	Description	Base Type
vi	Instrument handle	ViSession

MeasurementValue	Pass the measurement value you obtain from one of the Read or Fetch functions.	ViReal64
------------------	--	----------

If the function generates one or more outputs, this section contains a second table for output parameters. It also has three columns. The left column is labeled Outputs and the middle column is labeled Description. The right column is labeled Base Type.

For example:

Outputs	Description	Base Type
IsOverrange	Returns whether the MeasurementValue is a valid measurement or an overrange condition. Valid Return values: True - overrange condition occurred. False - valid measurement.	ViBoolean

Defined Values for a Parameter: (Optional)

Occasionally, function parameters are enumerations without a corresponding attribute. In this case, the enumeration shall be described using a table of the form used to describe the defined values for an enumerated attribute. The title of this part shall use the actual name of the parameter in place of the words "a Parameter".

Return Values:

Defines the possible completion codes for the function. This section shall contain at least the text "The *IVI-3.2: Inherent Capabilities Specification* defines general status codes that this function can return."

If the function can return additional instrument class specific values, it shall contain the text "The *IVI-3.2: Inherent Capabilities Specification* defines general status codes that this function can return. The table below specifies additional instrument class-defined status codes for this function."

Completion Codes	Description
Waveform In Use	The function generator is currently configured to produce the specified waveform or the waveform is part of an existing sequence.

Compliance Notes: (Optional)

The General Requirements section defines the general rules an IVI specific driver must follow to be compliant with a function. This section specifies additional compliance requirements and exceptions that apply to a particular function.

If the function does not contain any additional compliance requirements, the section shall be omitted.

15.5 Attribute ID Definitions Layout

This section shall contain a table titled, "<ClassName> Attribute ID Values", with two columns. The left column is titled "Attribute Name" and under it are all the attributes defined in the instrument class specification. The right column is titled "ID Definition" and under it is a numerical value for the attribute. This value may be in terms of other defined values.

15.6 Attribute Value Definitions Layout

This section shall contain a table for each attribute which has defined values. These tables each have four columns. The left column is titled "Value Name" and under it are all the values defined in the instrument class specification. The second column is titled "Language" and under it are entries for every languages of interest to IVI. The third column is titled "Identifier" and under it are the identifiers used in the various languages. The right column is titled "Actual Value" and under it is a numerical value for the value. This value may be in terms of other defined values.

For example,

Current Limit Behavior

<i>Value Name</i>	<i>Language</i>	<i>Identifier</i>	<i>Actual Value</i>
Current Regulate	C	IVIDCPWR_VAL_CURRENT_REGULATE	0
	COM	IviDCPwrCurrentRegulate	0
Current Trip	C	IVIDCPWR_VAL_CURRENT_TRIP	1
	COM	IviDCPwrCurrentTrip	1
Current Limit Behavior Class Ext Base	C	IVIDCPWR_VAL_CURRENT_LIMIT_BEHAVIOR_CLASS_EXT_BASE	100
	COM	N/A	
Current Limit Behavior Specific Ext Base	C	IVIDCPWR_VAL_CURRENT_LIMIT_BEHAVIOR_SPECIFIC_EXT_BASE	1000
	COM	N/A	

15.7 Function Parameter Value Definitions Layout

This section contains a subsection for each function which has one or more parameters which have defined values. The title of the subsection is the generic name of the function.

Each subsection shall contain a table for each parameter in the function which has defined values. The table has the same format at the tables in Attribute Value Definitions Layout with an additional top row containing Parameter and the parameter name.

For example,

Configure Output Range

Parameter: RangeType

COM Enumeration Name: IviDcPwrRangeTypeEnum

<i>Value Name</i>	<i>Language</i>	<i>Identifier</i>	<i>Actual Value</i>
Current	C	IVIDCPWR_VAL_RANGE_CURRENT	0
	COM	IviDCPwrRangeCurrent	0
Voltage	C	IVIDCPWR_VAL_RANGE_VOLTAGE	1
	COM	IviDCPwrRangeVoltage	1
Configure Output Range Class Ext Base	C	IVIDCPWR_VAL_RANGE_TYPE_CLASS_EXT_BASE	500
	COM	N/A	
Configure Output Range Specific Ext Base	C	IVIDCPWR_VAL_RANGE_TYPE_SPECIFIC_EXT_BASE	1000

	COM	N/A	
--	-----	-----	--

15.8 Error and Completion Code Value Definitions Layout

Defines all the error and completion that the instrument class specifies. This section contains two tables. The first table lists the error name as it is used throughout the instrument class specification, a more complete description of the error, the identifiers used in languages of interest to IVI with an associated value. The table has this form:

Table 15-4. IviScope Error and Completion Codes

<i>Error Name</i>	<i>Description</i>		
	<i>Language</i>	<i>Identifier</i>	<i>Value(hex)</i>
Invalid Waveform Element	One of the elements in the waveform array is invalid.		
	C	IVISCOPE_WARN_INVALID_WFM_ELEMENT	0x3FFFA2001
	COM	S_IVISCOPE_INVALID_WFM_ELEMENT	0x80042001
Channel Not Enabled	Specified channel is not enabled.		
	C	IVISCOPE_ERROR_CHANNEL_NOT_ENABLED	0xBFFFA2001
	COM	E_IVISCOPE_CHANNEL_NOT_ENABLED	0x80042001
Unable To Perform Measurement	Unable to perform desired measurement operation.		
	C	IVISCOPE_ERROR_UNABLE_TO_PERFORM_MEASUREMENT	0xBFFFA2002
	COM	E_IVISCOPE_UNABLE_TO_PERFORM_MEASUREMENT	0x80042002
Max Time Exceeded	Maximum time exceeded before the operation completed.		
	C	IVISCOPE_ERROR_MAX_TIME_EXCEEDED	0xBFFFA2003
	COM	E_IVISCOPE_MAX_TIME_EXCEEDED	0x80042003
Invalid Acquisition Type	Invalid acquisition type		
	C	IVISCOPE_ERROR_INVALID_ACQ_TYPE	0xBFFFA2004
	COM	E_IVISCOPE_INVALID_ACQ_TYPE	0x80042004

The second table defines the format of the message string associated with the error. In C, this string is returned by the Error Message function. In COM, this string is the description contained in the ErrorInfo object. The second table has this form:

Note: In the description string table entries listed below, %s is always used to represent the component name.

Table 15-5. IviScope Error Message Strings

Name	Message String
Invalid Waveform Element	“%s: Invalid waveform element”
Channel Not Enabled	“%s: Channel not enabled”
Unable To Perform Measurement	“%s: Unable to perform measurement”
Max Time Exceeded	“%s: Maximum time exceeded”
Invalid Acquisition Type	“%s: Invalid acquisition type”

15.9 Hierarchies

Each Hierarchies section shall contain the subsections: COM Hierarchy, C Function Hierarchy, and C Attribute Hierarchy.

15.9.1 COM Hierarchy

This section shall contain a table with three columns showing which properties and methods are in each interface. The sample table shown here lists the inherent methods and properties though individual instrument class specifications shall not include them. Do include a paragraph of the form:

“The full <ClassName> COM Hierarchy includes the Inherent Capabilities Hierarchy as defined in Section 4.1, *COM Inherent Capabilities of IVI-3.2: Inherent Capabilities Specification*. To avoid redundancy, the Inherent Capabilities are omitted here.”

The table shall have the form:

Table n+4-1. <ClassName> COM Hierarchy

COM Interface Hierarchy	Generic Name	Type
Close	Close	M
DriverOperation		
Cache	Cache	P
ClearInterchangeWarnings	Clear Interchange Warnings	M
DriverSetup	Driver Setup	P
GetNextCoercionRecord	Get Next Coercion Record	M
GetNextInterchangeWarning	Get Next Interchange Warning	M
InterchangeCheck	Interchange Check	P
InvalidateAllAttributes	Invalidate All Attributes	M
LogicalName	Logical Name	P
QueryInstrumentStatus	Query Instrument Status	P
RangeCheck	Range Check	P
RecordCoercions	Record Value Coercions	P
ResetInterchangeCheck	Reset Interchange Check	M
IoResourceDescriptor	Resource Descriptor	P
Simulate	Simulate	P
Identity		
Description	Component Description	P
Locator	Component Locator	P
Prefix	Component Prefix	P
Revision	Component Revision	P
Vendor	Component Vendor	P
InstrumentFirmwareRevision	Instrument Firmware Revision	P
GroupCapabilities	Class Group Capabilities	P
InstrumentManufacturer	Instrument Manufacturer	P

COM Interface Hierarchy	Generic Name	Type
InstrumentModel	Instrument Model	P
SpecificationMajorVersion	Component Class Spec Major Version	P
SpecificationMinorVersion	Component Class Spec Minor Version	P
SupportedInstrumentModels	Supported Instrument Models	P
Initialize	Initialize With Options	M
Initialized	Initialized	P
Utility		
Disable	Disable	M
ErrorQuery	Error Query	M
LockSession	Lock Session	M
Reset	Reset	M
ResetWithDefaults	Reset With Defaults	M
SelfTest	Self Test	M
UnlockSession	Unlock Session	M

This section shall also contain two addition subsections: Interfaces and Interface Reference Properties.

15.9.1.1 Interfaces

This section describes all the COM interfaces exposed by the IVI-COM driver. It contains a subsection for each interface. It includes a table of the form:

Table n+4-6. <ClassName> Interface GUIDs

Interface	GUID
I<ClassName>	{00000000-0000-0000-0000-000000000000}
I<ClassName><Itf1>	{00000000-0000-0000-0000-000000000000}
I<ClassName><Itf2>	{00000000-0000-0000-0000-000000000000}
I<ClassName><Itf3>	{00000000-0000-0000-0000-000000000000}

The GUIDs in the table are the actual GUID associated with the interface in the IDL.

15.9.1.2 COM Interface Reference Properties

This section describes all the interface reference properties used to navigate the COM hierarchy. It contains a subsection, 4th level, for each interface reference property. Each COM Interface Reference Property section shall contain the following subsections:

Interface Reference Table:

A table with two columns with the following titles:

Data Type:

Specifies the data type of the attribute. It is always a pointer to interface defined in the previous section.

Access:

Specifies the kind of access the user has to the attribute. It is always RO.

For example,

Data Type	Access
IIVIvDCPwrOutputs*	RO

COM Property Name

The name used by the user to gain access to this interface.

Description

Describes the property.

15.9.1.3 COM Category

This section specifies the COM Category and Category ID (CATID). For example,

The IviFgen class COM Category shall be “IviFgen”, and the Category ID (CATID) shall be {47ed5156-a398-11d4-ba58-000064657374}.

15.9.2 C Function Hierarchy

This section shall contain a table. The table shall follow the form of **Table 13-1** Prefix Function Hierarchy. Section 13.1 describes the contents of this table.

The table shall not include any of the inherent functions. Instead the section shall contain a paragraph of the form:

“The <ClassName> class function hierarchy is shown in the following table. The full <ClassName> C Function Hierarchy includes the Inherent Capabilities Hierarchy as defined in Section 4.2, *C Inherent Capabilities of IVI-3.2: Inherent Capabilities Specification*. To avoid redundancy, the Inherent Capabilities are omitted here.”

15.9.3 C Attribute Hierarchy

This section shall contain a single table with two columns. The sample table shown here lists the inherent attributes though individual instrument class specifications shall not include them. Do include a paragraph of the form:

The <ClassName> class attribute hierarchy is shown in the following table. The full <ClassName> C Attribute Hierarchy includes the Inherent Capabilities Hierarchy as defined in Section 4.2, *C Inherent Capabilities of IVI-3.2: Inherent Capabilities Specification*. To avoid redundancy, the Inherent Capabilities are omitted here.

The table has the form:

Table n+4-5. <ClassName> C Attributes Hierarchy

Category or Generic Attribute Name	C Defined Constant
<i>Inherent IVI Attributes</i>	
<i>User Options</i>	
Range Check	<CLASS_NAME>_ATTR_RANGE_CHECK

Table n+4-5. <ClassName> C Attributes Hierarchy

Category or Generic Attribute Name	C Defined Constant
Query Instrument Status	<CLASS_NAME>_ATTR_QUERY_INSTRUMENT_STATUS
Cache	<CLASS_NAME>_ATTR_CACHE
Simulate	<CLASS_NAME>_ATTR_SIMULATE
Record Value Coercions	<CLASS_NAME>_ATTR_RECORD_COERCIONS
Interchange Check	<CLASS_NAME>_ATTR_INTERCHANGE_CHECK
<i>Class Driver Identification</i>	
Class Driver Description	<CLASS_NAME>_ATTR_CLASS_DRIVER_DESCRIPTION
Class Driver Prefix	<CLASS_NAME>_ATTR_CLASS_DRIVER_PREFIX
Class Driver Vendor	<CLASS_NAME>_ATTR_CLASS_DRIVER_VENDOR
Class Driver Revision	<CLASS_NAME >_ATTR_CLASS_DRIVER_REVISION
Class Driver Class Spec Major Version	<CLASS_NAME >_ATTR_CLASS_DRIVER_CLASS_SPEC_MAJOR_VERSION
Class Driver Class Spec Minor Version	<CLASS_NAME >_ATTR_CLASS_DRIVER_CLASS_SPEC_MINOR_VERSION
<i>Driver Identification</i>	
Specific Driver Description	<CLASS_NAME >_ATTR_SPECIFIC_DRIVER_DESCRIPTION
Specific Driver Prefix	<CLASS_NAME >_ATTR_SPECIFIC_DRIVER_PREFIX
Specific Driver Locator	<CLASS_NAME >_ATTR_SPECIFIC_DRIVER_LOCATOR
Specific Driver Vendor	<CLASS_NAME >_ATTR_SPECIFIC_DRIVER_VENDOR
Specific Driver Revision	<CLASS_NAME >_ATTR_SPECIFIC_DRIVER_REVISION
Specific Driver Class Spec Major Version	<CLASS_NAME >_ATTR_SPECIFIC_DRIVER_CLASS_SPEC_MAJOR_VERSION
Specific Driver Class Spec Minor Version	<CLASS_NAME >_ATTR_SPECIFIC_DRIVER_CLASS_SPEC_MINOR_VERSION
<i>Driver Capabilities</i>	
Supported Instrument Models	<CLASS_NAME >_ATTR_SUPPORTED_INSTRUMENT_MODELS
Class Group Capabilities	<CLASS_NAME >_ATTR_GROUP_CAPABILITIES
<i>Instrument Identification</i>	
Instrument Manufacturer	<CLASS_NAME >_ATTR_INSTRUMENT_MANUFACTURER
Instrument Model	<CLASS_NAME >_ATTR_INSTRUMENT_MODEL
Instrument Firmware Revision	<CLASS_NAME >_ATTR_INSTRUMENT_FIRMWARE_REVISION
<i>Advanced Session Information</i>	
Logical Name	<CLASS_NAME >_ATTR_LOGICAL_NAME
I/O Resource Descriptor	<CLASS_NAME >_ATTR_IO_RESOURCE_DESCRIPTOR

Table n+4-5. <ClassName> C Attributes Hierarchy

Category or Generic Attribute Name	C Defined Constant
Driver Setup	<CLASS_NAME >_ATTR_DRIVER_SETUP

15.10 Appendix A: IVI Specific Driver Development Guidelines Layout

Each IVI Specific Driver Development Guidelines appendix shall contain the subsections:

- A.1 Introduction
- A.2 Disabling Unused Extension Groups
- A.3 through n Special Considerations for ... (optional)

Section A.2 contains an entry for each extension group. The special consideration sections contain information about any topic of interest to IVI driver writers. For example, how to implement instrument class required features for instruments that do not follow the model assumed in the specification.

15.11 Appendix B: Interchangeability Checking Rules Layout

Each Interchangeability Checking Guidelines appendix shall contain the subsections:

- B.1 Introduction
- B.2 When to Perform Interchangeability Checking
- B.3 Interchangeability Checking Rules

Section B.3 shall contain the names of every base and extension in bold on a separate. Each of these names shall be followed by one or more paragraphs describing the driver's behavior for that group.

15.12 Appendix C: ANSI C Include File

Include a template include file that contains prototypes and definitions for all functions, attributes, attribute values, and error codes that the IVI instrument class specification defines. The definitions necessarily do not represent a complete interface for an instrument class compliant driver. IVI driver writers will include the declarations appropriate for their drivers in the actual driver include file.

15.13 Appendix D: COM IDL File

Include the IDL that represents what was described in the instrument class specification. IVI driver writers can expect the IDL to compile into a usable type library.

16. Expressing Tone

Grammatically, auxiliaries are added before verbs to express tone. Be precise when using auxiliaries so the reader is never confused about the intent of the specification. Omitting an appropriate auxiliary can be as confusing as the wrong one.

16.1 Requirement

The auxiliaries shown in Table 16-1 shall be used to indicate requirements strictly to be followed in order to conform to the specification and from which no deviation is permitted.

Table 16-1 Requirement

Auxiliary	Equivalent expressions for use in exceptional cases
shall	is to is required to it is required that has to only ... is permitted it is necessary
shall not	is not allowed [permitted] [acceptable] [permissible] is required to be not is required that ... be not is not to be

Do not use “must” as an alternative for “shall”. Do not use “may not” instead of “shall not” to express a prohibition. To express a direct instruction, for example referring to steps to be taken in a test method, use the imperative mood in English.

16.2 Recommendation

The auxiliaries shown in Table 16-2 shall be used to indicate that among several possibilities one is recommended as particularly suitable, without mentioning or excluding others, or that a certain course of action is preferred but not necessarily required, or that (in the negative form) a certain possibility or course of action is deprecated but not prohibited.

Table 16-2 Recommendation

Auxiliary	Equivalent expressions for use in exceptional cases
should	it is recommended that ought to
should not	it is not recommended that ought not to

16.3 *Permission*

The auxiliaries shown in Table 16-3 are used to indicate a course of action permissible within the limits of the specification.

Table 16-3 Permission

Auxiliary	Equivalent expressions for use in exceptional cases
may	is permitted is allowed is permissible
need not	it is not required that no ... is required

Do not use “possible” or “impossible” in this context. Do not use “can” instead of “may” in this context.

“May” signifies permission expressed by the standard, whereas “can” refers to the ability of a user of the standard or to a possibility open to him.

16.4 *Possibility and Capability*

The auxiliaries shown in Table 16-4 are used for statements of possibility and capability, whether material, physical or causal.

Table 16-4 Possibility and Capability

Auxiliary	Equivalent expressions for use in exceptional cases
can	be able to there is a possibility of it is possible to
cannot	be unable to there is no possibility of it is not possible to