IVI-3.5: Configuration Server Specification

June 29, 2021

Revision 2.5

Important Information

The IVI Configuration Server Specification (IVI-3.5) 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`.

The IVI Foundation wants to receive your comments on this specification. You can contact the Foundation through the web site at `www.ivifoundation.org`.

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.

LIST OF FIGURES

Configuration Server Specification
Revision History

This section is an overview of the revision history of the IVI Configuration Server specification.

Table 1-1. IVI-3.5 Revisions

Revision Number	Date of Revision	Revision Notes
Revision 0.1	??	Original draft.
Revision 0.2	May 23, 2001	Bring up to date.
Revision 0.3	July 31, 2001	First version to reflect UML diagrams for iteration 6 of the Configuration Server.
Revision 0.4	August 27, 2001	Add Section 3 and incorporate design changes
Revision 0.5	September 10, 2001	Reflect changes discussed at Sept. 2001 IVI Foundation meeting
Revision 0.6	November 1, 2001	Reflect changes made in telephone conferences after the Sept 2001 IVI meeting.
Revision 0.7	November 30, 2001	Reflect changes made in telephone conferences after the Nov. 1, 2001 WG telephone conference.
Revision 0.8	December 18, 2001	Reflects changes made through the December, 2001 IVI Meeting and immediately after.
Revision 0.9	February 7, 2002	Reflects changes made through the February 7, 2002 telephone conference. Sections 1-23 are completely reviewed, ready for C API to be added.
Revision 0.91	March 20, 2002	C API was added in previous version. General cleanup..
Revision 0.92	April 16, 2002	New section for general C functions. Return codes sections added.
Revision 1.0vc1	April 29, 2002	Version for initial two week review.
Revision 1.0vc2	May 15, 2002	Changes made from two week review.
Revision 1.0vc	June 11, 2002	Changes made after two week review.
Revision 1.0	November 7, 2002	Aproved.
Revision 1.4	August 23, 2003	Modified sections describing configurable initial settings.
Revision 1.5	January 12, 2007	Clarification of permissible values for ModulePath in Section 10.3.1

Revision 1.6	February 8. 2008 (Approved)	Add the following new properties to the Software Module interface:
		`AssemblyQualifiedClassName`
		`ModulePath32`
		`ModulePath64`
		Add Vista as a supported OS.
		Remove the IDL and XML schema appendices.

Revision 1.6	March, 2008	Editorial change to update the IVI Foundation contact information in the Important Information section to remove obsolete address information and refer only to the IVI Foundation web site.
Revsion 1.7	November 17, 2008	Variety of editorial and minor changes related to the 64-bit implementations.

Table 1-1. IVI-3.5 Revisions

Revision 1.8	March 31, 2009	Updated references to IVI-3.1 Installation Requirements section to refer to IVI-3.17.
Revision 2.0	June 9, 2010	Incorporated IVI.NET
Revision 2.1	January 18, 2012	Minor change in Sections 2.9.3 and 27 to avoid conflict between physical and virtual names.
Revision 2.2	March 6, 2013	Minor changes to add Windows 8 as a supported OS
Revision 2.3	October 22, 2013	Minor Change in Section 11.3.1 to add the term “qualified repeated capability identifier”.
Revision 2.3	August 6, 2015	Editorial change to add Windows 10 as a supported operating system
Revision 2.4	June 7, 2016	Minor change to remove support for Windows Vista
Revision 2.4	May 19, 2017	Editorial change to add “IVI.NET” as an allowable published API type.
Revision 2.4	June 7, 2019	Editorial change to section 1.1 to clarify that there are multiple Config Server libraries distributed with the IVI Shared Components.
Revision 2.5	June 29, 2021	Minor change: Specify the native .NET Configuration Server API. Correct various formatting and typo errors.

1. Overview of the IVI Configuration Server Specification

This document describes IVI Configuration Servers that are provided by the IVI Foundation. Following the introduction, the general capabilities of the system are listed. The top-level architecture design is given with a component diagram and terminologies used. Capabilities of major components in the architecture are also described. Detailed descriptions of all the interface properties and functions follow. A sample use scenario diagram is then given. Next, the requirements on clients that use the IVI Configuration Server in a system are listed.

1.1 Introduction

IVI Configuration Servers are the run-time libraries that are responsible for providing system database services to IVI based measurement system applications. Specifically, they provide system initialization and configuration information. The IVI Configuration Servers are used by several of the IVI compliant modules. For instance, a Configuration Server indicates which physical instrument and IVI driver will be used by a particular application to provide a particular measurement capability.

Since a typical system intermixes instruments and drivers from multiple vendors this system configuration service needs to be accessed in a vendor independent fashion. Therefore, IVI Configuration Servers are IVI shared components (that is, the code is owned by the IVI Foundation). IVI Configuration Servers are provided by the IVI Foundation because the architecture requires that the Configuration Servers installed on any system behave consistently with each other and with this specification. By using one of the IVI Foundation Configuration Servers, customers eliminate potential conflicts from custom implementations that diverge from the specification.

An IVI Configuration Server is an executable library that works with one or more XML configuration stores (databases). Together, they include the following basic components:

- The physical database (known as the configuration store). A physical configuration store is a single XML file. APIs are available to read and write the data to arbitrary files, thus providing complex applications with the ability to directly manage system configurations.
- The API (and its implementation) used to read information from the configuration store(s). The IVI modules typically use this API when they are instantiated and configured.
- The API (and its implementation) to write information to the configuration store(s). This API is typically used by GUI or other applications that set up the initial configuration.
- The API (and its implementation) used to bind an instance of the Configuration Server code to a particular copy of the configuration information stored on a system. This includes appropriate algorithms for gaining access to the master configuration store.

1.2 Typical Use Scenario of the Configuration Server

The following example illustrates the typical operations conducted with an IVI Configuration Server.

1. Various instrument drivers are installed on the system. As each instrument driver is installed on the system, its installation script makes entries in the configuration store that indicate the location of the driver, its ProgID, a list of instruments it supports, and the interfaces it provides to its client. This entry is known as a *SoftwareModule* because it describes the software module (in this case, an instrument driver) that was installed. Software module developers determine what happens at installation time, and are the primary actors at this step.

2. A user configuring the system makes entries in the configuration store that indicate a logical name for each instrument service they will be using. Then, the user associates a specific instrument and driver with each logical name. Note that the physical instrument is primarily identified by its I/O address. The driver is entered as a reference to a *SoftwareModule* entry that was created by the driver installation. In addition, the user may provide information regarding the default behavior of the instrument. This step will typically be completed with the aid of a configuration utility, however a

configuration utility is beyond the scope of the IVI specifications. Users determine how the software modules are configured, and are the primary actors at this step.

3. For COM software modules, when the user's application runs, it instantiates the IVI-COM Session Factory (refer to IVI-3.6, *COM Session Factory Specification*). The user application then calls *CreateDriver()* passing it the logical name they defined in the configuration step. The factory then instantiates the software module and configures it based on the entries provided in step 2. The user is the primary actor at this step.

4. For .NET software modules, the user's application calls *Ivi.<ClassName>Create* or *Ivi.Driver.Create* passing it the logical name they defined in the configuration step. The Create method then instantiates the software module and configures it based on the entries provided in step 2. The user is the primary actor at this step.

The benefit of this use scenario is that the user's program is entirely de-coupled from the configuration information. It is therefore possible to modify the configuration information provided in step 2 above without ever modifying the actual program that invokes and uses the driver. The benefit is that an instrument with a class-compliant driver can replace another class-compliant driver in an existing system, with no code changes made to the user application program.

This pattern of associating configuration information with a logical name, and thus allowing a system to be re-configured without code changes is known as an abstract factory pattern and has other applications within the IVI architecture. For instance, IVI-MSS role control modules make similar use of the IVI Configuration Server.

An IVI Configuration Server also allows users to associate arbitrary information with software modules and logical names. This can be useful when there is additional configuration information that is needed by the application. The IVI Configuration Server defines several fields specifically for use with instrument driver sessions.

1.2.1 Repeated Capabilities

In many instruments there are capabilities that are duplicated either identically or very similarly across the instrument. Such capabilities are called repeated capabilities. The IVI class-compliant APIs represent repeated capabilities by a parameter that indicates which instance of the duplicate capability this function is intended to access. The IVI C APIs include this parameter as an additional parameter to function calls. The IVI COM APIs may do the same, or may also use this parameter as an index into a repeated capability collection.

An IVI Configuration Server provides a way for software modules to publish the functionality that is duplicated and the strings that the software module recognizes to access the repeated capabilities. An IVI Configuration Server also provides a way for the client to supply aliases for the *physical identifiers* recognized by the drivers.

Since many instruments have numerous instances of repeated capabilities, an IVI Configuration Server provides a way to represent the repeated capabilities as a range of identifiers instead of many individual identifiers.

One repeated capability may also be related to another repeated capability in a hierarchical parent/child relationship. The child repeated capabilities in these relationships are called *nested repeated capabilities*. An IVI Configuration Server provides a way to model these relationships.

1.3 References

Several other documents and specifications are related to this specification. These other related documents are the following:

- IVI-3.1: Driver Architecture Specification
- IVI-3.2: Inherent Capabilities Specification

- IVI-3.4: API Style Guide
- IVI-3.6: COM Session Factory Specification
- IVI-3.17 Installation Requirements Specification

1.4 Definitions of Terms and Acronyms

Terms of general interest are defined in IVI-5.0: Glossary.

Symmetrical Repeated Capability – A repeated capability where each instance of a repeated capability has identical capabilities to all of the other instances.

1.5 Implementation

The IVI Foundation supplied implementations of the IVI Configuration Server and IVI Session Factory are available from the IVI Foundation web site. These are packaged with the other IVI Foundation shared components as part of the shared component installation package.

There are currently three shipping implemenations of the IVI Configuration Server, a native IVI-C implementation, a native IVI.NET implementation, and an IVI-COM implementation that includes a .NET primary interop assembly. Note that in the rest of the spec, “the configuration server” may refer to any implementation, or to all of them collectively.

2. IVI Configuration Server Design

The IVI Configuration Server is based on an object oriented UML (Unified Modeling Language) design. The Configuration Server data is stored as an XML configuration store file that closely follows the design of the Configuration Server.

2.1 UML Design

The IVI Configuration Server design is most easily understood by considering an implementation-independent class diagram for the API. The XML data structure closely follows the structure of the API. The UML class diagram is shown in Figure 2-1.

In the diagram, a rectangle represents a class. A dotted line indicates class inheritance, with the triangle pointing to the inherited interface. Note that IviConfigComponent and IviDataComponent are both abstract base classes, and are never implemented directly by the Configuration Server. Although it is a base class, IviSession is not abstract, and is directly implemented.

The dashed and solid lines are references from the class at the tail of the arrow to the class at the head of the arrow. For each reference, it is assumed that the class at the tail of the arrow contains a reference property (a property that returns a reference to another object) named by the text at the head of the arrow.

Collection classes are implied by the UML diagram wherever there is a relationship (indicated by a solid line) with an ordinality of 0..* or 1..* at the head of the arrow. For example, the IviLogicalNames collection class is implied by the Logical Names relationship between IviConfigStore and IviLogicalName. The IviConfigStore class includes a reference property named “LogicalNames” which references an IviLogicalNamesCollection object, which manages a collection of zero or more references to IviLogicalName objects. The Name property uniquely identifies an object in a collection. Refer to section 4, *Collections*, for more information about Configuration Server collections.

A heavy dashed line represents a reference to a global collection of all of the objects in a global class (refer to the next section). A solid line represents other references.

2.2 Types of Classes and Objects

Every instance of an IVI Configuration Server has exactly one instance of the IviConfigStore class. This object is instantiated by users who request an instance of the IVI Configuration Server. Users can navigate to all of the other objects in the configuration store from this object.

There are six “global” classes – IviLogicalName, IviHardwareAsset, IviSoftwareModule, IviPublishedAPI, IviSession, and IviDriverSession. Objects in these classes may be referenced from any object in the configuration server that implements a corresponding reference property. All of the objects of a global class are unique (by Name), except Published APIs, within the entire Configuration Server. Published APIs are differentiated by Name, Version, and Type. For example, all of the software module objects in the configuration server have a unique Name.

There are six global collections in the configuration store, one for each global class. All of the objects of a global class are part of the corresponding global collection. The global collections are referenced by the main IviConfigStore object.

Although an object in a global collection may be referenced by other objects, it exists independently of those other objects. When the referencing object is deleted, the global object is not necessarily deleted. In fact, with one exception. it may only be deleted when there are no outstanding references to it in the configuration server. For example, a session may reference a hardware asset, but the hardware asset is not necessarily deleted when the session is deleted. The hardware asset may only be deleted from the global collection when there are no sessions or driver sessions that refer to it.

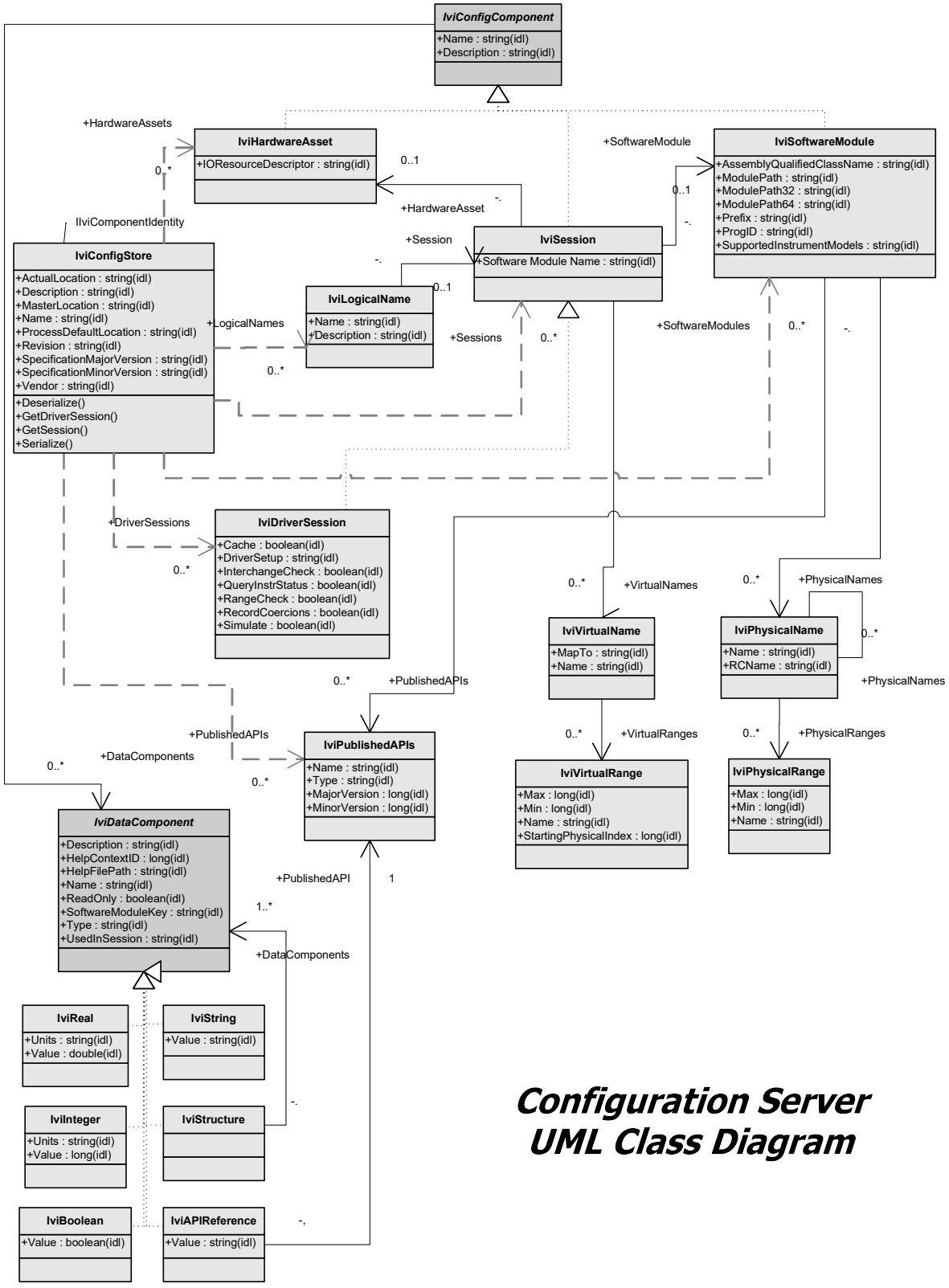

Figure 2-1 IVI Configuration Server UML Class Diagram

The "exception" to this rule is the relationship from session to software module. A software module may be deleted when there are still sessions that reference it. In this case, the session cannot reference an IVI Software Module object, but it does "remember" the name of the software module that it referenced, and if

a software module with the same name is ever re-added to the configuration store, a reference from the session to that software module is recreated.

The other classes (including other collection classes) are "contained" classes. Contained objects in these classes are not global, and are unique and meaningful only in the context of a "containing" object that contains a reference to a collection of the "contained" objects. Contained objects do not exist independently of the containing object, and are deleted when the containing object is deleted. For example, all of the Data Component objects referenced by a software module are unique to that software module, and must be deleted when the software module is deleted. This implies that software modules cannot share Data Component objects – each software module requires a duplicate Data Component for identical configuration variables.

Note that the Software Module class references a collection of Published APIs. This collection is not a global collection. It is "contained" in the software module that references it, and is deleted when the software module is deleted. However, the Published API objects that are referenced by the collection are not deleted, since they are also in the global Published API collection.

2.3 Notation

In the descriptions of each class that follow,

- API methods are identified as such.
- API properties that do not reference another object are followed by the basic type (Boolean, string, long, or double) and an indication of whether the property is read-only (R/O), write-only (W/O), or both (R/W). If the property is a string, Optional indicates that the string may be empty. It is assumed that longs, doubles, and Booleans must always have a valid value. All strings must be legal XML strings, since the configuration store is an XML file.
- API properties that reference an object are followed by the type of the object, and an indication of whether the reference is optional (0..1) or required (1). If the reference is optional, the property returns a NULL pointer if there is no reference.
- API properties that reference a collection object are followed by the type of the collection, and an indication of whether the collection may be empty (0..*) or not (1..*).

2.4 IVI Configuration Store

The IviConfigStore class contains the following references, properties, and methods.

- ***Driver Sessions*** (Collection of IviDriverSession, 0..*) – The global collection of references to all of the IVI Driver Session objects in the configuration store.
- ***Hardware Assets*** (Collection of IviHardwareAsset, 0..*) – The global collection of references to all of the IVI Hardware Asset objects in the configuration store.
- ***Logical Names***(Collection of IviLogicalName, 0..*) – The global collection of references to all of the IVI Logical Name objects in the configuration store.
- ***Published APIs*** (Collection of IviPublishedAPI, 0..*) – The global collection of references to all of the IVI Published API objects in the configuration store.
- ***Sessions*** (Collection of IviSession, 0..*) – The global collection of references to all of the IVI Session objects in the configuration store.
- ***Software Modules***(Collection of IviSoftwareModule, 0..*) – The global collection of references to all of the IVI Software Module objects in the configuration store.
- ***Actual Location*** (String, optional, R/O) – The full pathname of the IVI configuration store file that was deserialized by the current instance of the configuration server.
- ***Description*** (String, required, R/W) – The description of the configuration store.
- ***Master Location*** (String, optional, R/O) – The full pathname of the master IVI configuration store file.
- ***Name*** (String, required, R/O) – The name of the running Configuration Server component.
- ***Process Default Location*** (String, required, R/W) – The full pathname of the IVI configuration store file to be used by the current process.
- ***Revision*** (String, required, R/O) – The revision of this version of the Configuration Server. Will match the version reported by the DLL's file version resource. Refer to Section 5.1, *IVI-COM Interface Versioning*, of *IVI-3.4: API Style Guide* for more details.

- ***Specification Major Version*** (Long, R/O) – The major version of the IVI Configuration Server specification to which this version of the Configuration Server complies.
- ***Specification Minor Version*** (Long, R/O) – The minor version of the IVI Configuration Server specification to which this version of the Configuration Server complies.
- ***Vendor*** (String, required, R/O) – "IVI Foundation, Inc."
- ***Deserialize()*** (method) – Deserializes an instance of a configuration store into the Configuration Server. Note that the native .NET API does not include Deserialize, but instead provides Load factory methods that instantiate the ConfigStore class and deserialize a configuration store in one method.
- ***Get Driver Session()*** (method) – Given a name, navigates the Configuration Server's logical name collection and driver session collection to find the corresponding IVI Driver Session.
- ***Get Session()*** (method) – Given a name, navigates the Configuration Server's logical name collection and session collection to find the corresponding IVI Session.
- ***Serialize()*** (method) – Serializes the configuration store data in Configuration Server memory out to a configuration store file. Note that the native .NET API does not include Serialize, but instead provides a similar Save method, so named to parallel the .NET Load factory methods.

2.5 IVI Configurable Components

The IVI Configuration Server is organized around a series of IVI Configurable Components – classes that inherit from the IVI Configurable Component abstract base class. This class allows the addition of custom properties to any class that inherits from it, as well as the common Name and Description properties. Although it is legitimate to define additional IVI Configurable Components, the IVI Foundation only specifies the semantics of three classes derived from IVI Configurable Component.

- IVI Software Module
- IVI Session (and IVI Driver Session through inheritance)
- IVI Hardware Asset

2.5.1 IVI Configurable Component

The IVI Configurable Component class contains the following references and properties:

- ***Data Components*** (Collection of IviDataComponent, 0..*) – An optional collection of references to additional properties that may be added to any configurable component by its owner.
- ***Name*** (String, required) – This is a human readable name for this component. Name may be any valid string, but is used as a key or index value in collections. Therefore, Name must be unique within collections of like objects.
- ***Description*** (String, optional) - This is a human readable description of this component. Description may be any valid string.

2.5.2 IVI Software Module

The IVI Software Module class contains information that describes a software component installed on the system. This component only contains information relevant to the installed software module, it does not contain information that is associated with a running instance.

IVI Software Module inherits from IVI Configurable Component. In addition to the properties inherited from IVI Configurable Component, the IVI Software Module class contains the following properties:

- ***AssemblyQualifiedClassName*** (String, required for IVI-.NET) – For IVI-.NET software modules, the assembly qualified class name.
- ***Module Path*** (String, required for IVI-C) – For IVI-C software modules (including IVI-C wrappers), the full pathname or simple filename of the software module DLL. When running in a native 32-bit context, ModulePath returns ModulePath32. When running in a native 64-bit context, ModulePath returns ModulePath64.
  For backwards compatibility with earlier versions of the Configuration Server, ModulePath sets ModulePath32 when running in a native 32-bit context. When running in a native 64-bit context, attempts to set ModulePath return a Not Supported error.
- ***Module Path32*** (String) – For IVI-C software modules, the full pathname or simple filename of the native 32-bit software module DLL. ModulePath32 is required for IVI-C 32-bit software modules, including IVI-C wrappers. ModulePath32 shall be the empty string for all other types of IVI drivers.

- ***Module Path64*** (String) – For IVI-C software modules, the full pathname or simple filename of the native 64-bit software module DLL. ModulePath64 is required for IVI-C 64-bit software modules, including IVI-C wrappers. ModulePath64 shall be the empty string for all other types of IVI drivers.
- ***Prefix*** (String, optional) – The prefix (IVI-C) or identifier (IVI-COM, IVI.NET) of the software module.
- ***ProgID***(String, required for IVI-COM) – For IVI-COM software modules, the version independent ProgID of the registered software module.
- ***Supported Instrument Models*** (String, optional) - A comma separated list of supported instrument models. Required for IVI specific instrument drivers.
- ***Published APIs*** (Collection of IviPublishedAPI, 0..*) – The collection of references to the IVI Published API objects implemented by the software module.
- ***Physical Names*** (Collection of IviPhysicalName, 0..*) – The collection of references to the IVI Physical Identifier objects implemented by the software module. This collection describes information about the repeated capabilities names implemented by the software module. Refer to Section 2.9, *Repeated Capabilities,* for more information.

2.5.3 IVI Session and IVI Driver Session

The IVI Session class describes how an instance of IVI Software Module will be configured. The IVI Driver Session class defines an additional set of properties for use by IVI instrument drivers. The Ivi Driver Session class inherits from the Ivi Session class.

IVI Session inherits from IVI Configurable Component. In addition to the properties inherited from IVI Configurable Component, the IVI Session class contains the following references:

- ***SoftwareModule*** (Reference to IviSoftwareModule, 0..1) – A reference to the IVI Software Module object that is being configured by this session.
- ***HardwareAsset*** (Reference to IviHardwareAsset, 0..1) – A reference to the IVI Hardware Asset object to be used by the configured software module.
- ***VirtualNames*** (Collection of IviVirtualName, 0..*) – The collection of references to the IVI Virtual Name objects defined by the session. This collection describes information about the repeated capabilities names that can be used as aliases by the software module. Refer to Section 2.9, *Repeated Capabilities* for more information.
- ***SoftwareModuleName*** (String, optional, R/O) – The name of the current or most recently referenced software module referenced by the Software Module property.

In addition, the IVI Driver Session class contains the following properties. Refer to IVI-3.2, Section 5 for exact details.

- ***Cache*** (Boolean) – If TRUE, drivers that support state caching will initially enable that feature. Default is FALSE.
- ***DriverSetup*** (String, optional) – The content of this string is dependent on the software module associated with the driver session. The software module knows how to interpret the string.
- ***InterchangeCheck*** (Boolean) – If TRUE, drivers that support interchangeability checking will initially enable that feature. Default is FALSE.
- ***QueryInstrumentStatus*** (Boolean) – If TRUE, drivers will initially enable querying instrument status. Default is FALSE.
- ***RangeCheck*** (Boolean) – If TRUE, drivers that support extended range checking will initially enable that feature. Default is FALSE.
- ***RecordCoercions*** (Boolean) – If TRUE, drivers that support recording of coercions will initially enable that feature. Default is FALSE.
- ***Simulate*** (Boolean) – If TRUE, drivers will initially enable simulation. Default is FALSE.

2.5.4 IVI Hardware Asset

The IVI Hardware Asset class contains the I/O Address of a particular hardware asset. The form of this address is dependent on the underlying I/O mechanism. For IVI instrument drivers, this will commonly be a VISA Resource Descriptor string.

It is valid for a session to refer to a hardware asset even if the asset is not physically at the address. However, a software module trying to use the hardware asset will be unable to establish communication with it.

IVI Hardware Asset inherits from IVI Configurable Component. In addition to the properties inherited from IVI Configurable Component, the IVI Hardware Asset class contains the following property:

- ***IOResourceDescriptor*** (String, optional) – The I/O Address of a particular hardware asset.

2.6 IVI Logical Name

The IVI Logical Name class provides the binding between the user's program and the configuration information stored in the IVI configuration store. The users program identifies which session to instantiate by passing the Name associated with a particular IVI Logical Name component to the IVI Session Factory. The IVI Session Factory instantiates a session based on the configuration in the IviSession that is referred to by the IVI Logical Name with the name specified by the user.

The IVI Logical Name class contains the following reference and property:

- ***Session*** (Reference to IviSession, 0..1) – A reference to an IVI Session object.
- ***Name*** (String, required) – The logical name.
- ***Description*** (String, optional) – A human readable description of this logical name. Description may be any valid string.

2.7 IVI Published API

A published API defines the syntax and partial semantics of a programming interface that can be used to accomplish a particular task. The semantics are specified in sufficient detail to describe the task to be done, but enough semantics are left unspecified to leave room for a reasonable variety of implementations.

Published APIs defined by the IVI Foundation have names that begin with "Ivi". For IVI drivers, examples of published APIs are the inherent interfaces (named IviDriver) and the class-compliant interfaces (named IviDmm, IviScope, etc.). The names of IVI driver published APIs are defined by the IVI Foundation. For IVI-MSS, published APIs are the defined MSS measurement interfaces. For example, a phase noise measurement API defined by the IVI Foundation might be called IviMssPhaseNoise.

Published APIs defined outside of the IVI Foundation shall not have names that begin with "Ivi". For example, a phase noise measurement API defined by a T&M company might be called "TMCoPhaseNoise", where "TMCo" is the T&M company that developed the API.

The IVI Published API class contains the following properties:

- ***Name*** (String, required) – The name of the published API.
- ***Type*** (String, required) – The syntactical type of the API. Predefined values are "IVI.NET" for IVI.NET compliant interfaces, "IVI-COM" for IVI-COM compliant interfaces, and "IVI-C" for IVI-C compliant interfaces. One of these values must be used for all IVI defined APIs. Other values may be used if appropriate for other types of APIs.
- ***MajorVersion*** (Long) – The major version of the published API. For APIs that are defined by IVI specifications, this is the major version of the IVI specification.
- ***MinorVersion*** (Long) – The minor version of the published API. For APIs that are defined by IVI specifications, this is the minor version of the IVI specification.

2.8 IVI Data Components

The IVI Data Component class provides a way to attach arbitrary data to the IVI Configurable Components – hardware assets, software modules, and sessions (including driver sessions).

Data components may be used for two basic purposes

- Configurable Initial Settings: Data components that define and configure initial settings for additional variables known to a software module and configured by sessions that reference the software module.
- Documentation: Data components that document the configurable component itself.

2.8.1 IVI Data Component

IVI Data Components are the means of customizing a configuration store's data structure. They are used to add custom properties to IVI Configurable Components.

There are six data component classes, each of which inherits from IVI Data Component. IVI Data Component is an abstract base class. This class has properties that describe the data component, as well as the common Name and Description properties. The six data component classes that inherit from IviDataComponent are:

- IVI Structure
- IVI Boolean
- IVI Real
- IVI Integer
- IVI String
- IVI API Reference

The IVI Data Component class contains the following properties:

- ***Name*** (String, required) – A human readable name for this component. Name may be any valid string, but is used as a key or index value in collections. Therefore, Name must be unique within collections of DataComponents.
- ***Description*** (String, optional) - A human readable description of this component. Description may be any valid string.
- ***HelpContextID*** (Long) – The context ID of the help topic for this data component.
- ***HelpFilePath*** (String, optional) – The fully qualified help file pathname for the help file in which the help topic for this data component may be found.
- ***ReadOnly*** (Boolean) - Indicates a restriction in the client's permission to change the value of this data component. The IVI Configuration Server attaches no significance to this property and does not enforce any protocol regarding write access to data components. This property is primarily guidance for configuration utilities.
- ***SoftwareModuleKey*** (String, optional) – A string that is meaningful to the software module that identifies the data component or type of data component to the software module.
- ***Type*** (String, optional) - Contains a string that indicates which type of IVI Data Component this is. It will accurately reflect the type of this component and will contain one of the following values, "Structure", "Boolean", "Real", "Integer", "String", or "APIReference".
- ***UsedInSession*** (String) – Indicates whether or not a data component associated with a software module must be copied (UsedInSession = "Required"), may be copied (UsedInSession = "Optional"), or may not be copied (UsedInSession = "None") to any associated sessions. When associated with a hardware asset, UsedInSession is always "None".

The values of some properties are determined at least partially by what type of configurable component the data component is associated with, and the purpose of the data component.

	Configurable Initial Settings	Documentation
Hardware Asset	N/A	UsedInSession = "None" SoftwareModuleKey = ""
Software Module	UsedInSession = "Required"\"Optional" ReadOnly = True	UsedInSession = "None" SoftwareModuleKey = ""
Session/Driver Session	UsedInSession = "Required"\"Optional" ReadOnly = False	UsedInSession = "None" SoftwareModuleKey = ""

2.8.2 IVI Structure

The IVI Structure data component references a collection of data components. This allows the creation of complex structures.

In addition to the properties inherited from IVI Data Component, the IVI Structure class contains the following property:

- ***DataComponents*** (Collection of IviDataComponent, 1..*) – A collection of references to the IVI DataComponents. This collection defines a child structure of data components. Using this class, hierarchies of data components can be defined.

2.8.3 IVI Boolean

This class provides Boolean data in data components.

In addition to the properties inherited from IVI Data Component, the IVI Boolean class contains the following property:

- ***Value*** (Boolean) – Boolean data associated with this data component. Must be TRUE or FALSE.

2.8.4 IVI Real

This class provides real data in data components.

In addition to the properties inherited from IVI Data Component, the IVI Real class contains the following properties:

- ***Value*** (Double) – Real data associated with this data component.
- ***Units*** (String, optional) – Units associated with the number.

2.8.5 IVI Integer

This class provides integer data in data components.

In addition to the properties inherited from IVI Data Component, the IVI Integer class contains the following properties:

- ***Value*** (Long) – Integer data associated with this data component.
- ***Units*** (String, optional) – Units associated with the number.

2.8.6 IVI String

This class provides string data in data components.

In addition to the properties inherited from IVI Data Component, the IVI String class contains the following property:

- ***Value*** (String, optional) – String data associated with this data component.

2.8.7 IVI API Reference

This class provides a way for data component structures to reference published APIs. The Name property identifies the instance of the published API known by the software module. This data component is designed for use by IVI Software Modules and IVI Sessions.

In addition to the properties inherited from IVIi Data Component, the IVI String class contains the following property:

- ***PublishedAPI*** (Reference to IviPublishedAPI, 1) – A reference to an IVI Published API object to be used by the configured software module.
- ***Value***(String,optional) – A logical name or session name. Value can be passed to GetSession() or GetDriverSession() in the Name parameter. A session reference is returned according to the semantics defined for GetSession() and GetDriverSession().

2.9 Repeated Capabilities

Standard ways of referencing repeated capabilities in IVI software module APIs are described in several sections of IVI-3.1, Driver Architecture Specification, and Section 12, *Repeated Capabilities*, of the *IVI-3.4: API Style Guide*. Most IVI instrument class APIs include some repeated capabilities.

2.9.1 Repeated Capabilities in the Configuration Server

The IVI Configuration Server provides a way for software modules to publish their repeated capabilities and the physical identifiers that a client may use to access them. Software Modules use the IviPhysicalName and IviPhysicalRange classes for this purpose.

The IVI Configuration Server also provides a way for clients to configure instances of software modules for the client's use. As part of this configuration, clients specify virtual identifiers that the software module will recognize as aliases for the physical identifiers provided by the software module. Clients use the IviVirtualName and IviVirtualRange classes for this purpose.

2.9.2 IVI Physical Name and IVI Physical Range

Software modules publish their repeated capabilities using the IVI Physical Name and IVI Physical Range classes. Together these classes specify

- The repeated capabilities implemented by the software module. (Determined from the RCName property of IVI Physical Range objects.)
- The repeated capability hierarchy implemented by the software module. (Determined from the PhysicalNameCollection referenced by the IVI Physical Name objects, which may be used recursively to represent a hierarchy.)
- A set of physical identifiers recognized by the software module. There is exactly one physical identifier for each instance of a repeated capability implemented by the software module. (Determined from the Name property of, and the PhysicalRangeCollection referenced by, the IVI Physical Range objects.)

An IVI Physical Name object defines one or more physical identifiers corresponding to one or more instances of a repeated capability. The IVI Physical Name class contains the following properties:

- ***RCName*** (String, required) – The name of a repeated capability. All IVI Physical Names within the same collection that have the same RCName contribute to the definition of that repeated capability.
- ***Name*** (String, required) – If there is no associated physical range, the physical identifier for an instance of a repeated capability of type RCName. If there are associated physical range(s), the prefix for a range of physical identifiers to which integers from the range(s) are appended. May be empty if there are associated physical ranges, but note that since Name is a key for the IVI Physical Name collection, only one Name per collection may be empty. To avoid conflicts with the use of the colon character as a separator, Name may not contain a colon.
- ***PhysicalNames*** (Collection of IviPhysicalName, 0..*) – The collection of the IVI Physical Name objects for instances of repeated capabilities that are nested under the current physical identifier.
- ***PhysicalRanges*** (Collection of IviPhysicalRange, 0..*) – The collection of integer ranges used to create a set of physical identifiers, as explained below.

When the Physical Ranges property is not NULL, it references a collection of IVI Physical Range objects. A Physical Range object defines a range of integers between a minimum and maximum. The integers are appended to the Name property of the IVI Physical Name object to form a set of physical identifiers. If an IVI Physical Name object references a collection of IVI Physical Ranges, Name may be an empty string. This allows the range of names to be purely numeric. Also, since IVI Physical Name refers to a collection of IVI Physical Ranges, several non-contiguous ranges may be referenced. For example, a physical identifier "C" that references a collection of physical ranges 0-75, 100-175, and 200-275 yields a set of physical identifiers C0-C75, C100-C175, and C200-C275.

When an IVI Physical Name references IVI Physical Range(s), and IVI Physical Name(s), the collection of IVI Physical Names (and therefore the implied hierarchy of repeated capabilities) referenced by the IVI Physical Name is assumed to be symmetrical – the same for every physical identifier in the set. In the

above example, if the physical identifier "C" referenced a collection of physical identifiers consisting of "X", "Y", and "Z", it would be assumed that every one of the channels C0-C75, C100-C175, and C200-C275 had exactly the same set of child physical identifiers "X", "Y", and "Z".

For non-symmetrical nested repeated capabilities, Physical Ranges cannot be used.

The IVI Physical Range class contains the following properties:

- ***Name*** (String, required) – A value that uniquely identifies an IVI Physical Range object in a collection. There is no other significance to the Name field in this class.
- ***Max*** (Long, required) – The maximum integer in the range.
- ***Min*** (Long, required) – The minimum integer in the range.

2.9.2.1 Nested Repeated Capabilities

The software module represents nested repeated capabilities by creating an instance of IviPhysicalName for the parent capability (in this example, the output power), which in turn references another instance of IviPhysicalName for the nested capabilities (in this example, the external trigger). The nesting of IviPhysicalNames can be arbitrarily deep.

2.9.2.2 Symmetrical and Asymmetrical Nested Capabilities

Multiple instances of the same repeated capability may or may not share the same child repeated capabilities.

If multiple instances of the same repeated capability share the same child repeated capabilities, the repeated capability tree is symmetrical. In this case, each IVI Physical Name object for that repeated capability references an identical collection of child IVI Physical Name objects.

If multiple instances of the same repeated capability do not share the same child repeated capabilities, the repeated capability tree is asymmetrical. In this case, each (parent) IVI Physical Name object for that repeated capability will reference a collection of (child) IVI Physical Name objects that describe the specific child repeated capabilities for that object, and these collections may differ from one parent to the next.

For example, if a driver models two displays, and both displays can display two traces, that part of the repeated capability hierarchy would be symmetrical. If a driver models two displays, and the first can display four traces, but the second can only display two, that part of the repeated capability hierarchy is asymmetrical.

2.9.2.3 Uniqueness of IVI Physical Names

The IVI Physical Name ***Name*** property is unique within a collection of IVI Physical Names. In situations where collections of physical names include names for more than one repeated capability, this implies that ***Name*** is unique across all of the repeated capabilities in the collection.

For example, the IVI SpecAn class includes two repeated capabilities, "Trace" and "Marker". Drivers that implement the IVI SpecAn API will create a software module configuration store entry that includes a pointer to a physical names collection. The physical names collections will include the physical names that the driver defines for traces and markers. Since the physical names for traces and markers are stored in the same physical names collection, the ***Name*** property must be unique across all traces ***and*** markers.

If a driver includes nested repeated capabilities, the nesting physical name entry contains a reference to a different, non-empty collection of nested (e.g. child) physical names. In this case, the names in the collection of nested physical names need only be unique within that collection.

For example, if a driver defines a "trace" repeated capability, and then defines a "display" nested repeated capability that is a child of "trace", the "display" repeated capability could use names that are also used in the "trace" repeated capability.

2.9.3 IVI Virtual Name and IVI Virtual Range

The IVI Virtual Name and IVI Virtual Range classes are used by the client to create aliases for physical identifiers and colon separated lists of physical identifiers that the software module can use in context to select a repeated capability.

The IVI Virtual Name class contains the following properties:

- ***MapTo*** (String, required) – The string that is substituted by the software module for Name whenever it is encountered in a repeated capability selector. The empty string is a legal value for this property only if the IVI Virtual Name object references a non-empty collection of IVI Virtual Range objects.
- ***Name*** (String, required) – If there is no associated physical range, a virtual identifier. If there are associated physical range(s), the prefix for a range of virtual identifiers to which integers from the range(s) are appended. May be empty if there are associated virtual ranges. To avoid conflicts with the use of the colon character as a separator, Name may not contain a colon.
- ***VirtualRanges*** (Collection of IviVirtualRange, 0..*) – The collection of integer ranges used to create a set of IVI Virtual Names, as explained below.

The IVI Virtual name cannot be the same as any physical name in the collection of physical names this virtual name is mapped to. For instance, if the collection of physical names contains ‘foo’, ‘bar’, and ‘baz’, the virtual name cannot have a name of ‘foo’, irrespective of the value of *MapTo*. This rule is not enforced by the IVI Configuration Server.

An IVI Virtual Name object can also reference a collection of IVI Virtual Range objects. An IVI Virtual Range object defines a range of integers between a minimum and maximum. The integers are appended to the Name property of the IVI Virtual Name object to form a set of virtual identifiers. If an IVI Virtual Name object references a collection of IVI Virtual Ranges, Name may be an empty string. This allows the range of names to be purely numeric. Also, since IVI Virtual Name refers to a collection of IVI Virtual Ranges, several non-contiguous ranges may be referenced. For example, a virtual identifier “C” that references a collection of Virtual Ranges 0-75, 100-175, and 200-275 yields a set of virtual identifiers C0-C75, C100-C175, and C200-C275.

When an IVI Virtual Name has IVI Virtual Range(s), the set of virtual identifiers maps to a set of physical identifiers created by appending a series of integers to the IVI Virtual Name object’s MapTo string. The series of integers is the same size as the virtual range, but starts at the IVI Virtual Range object’s Starting Physical Index.

The IviVirtualRange class contains the following properties:

- ***Name*** (String, required) – A value that uniquely identifies an IVI Virtual Range object in a collection. There is no other significance to the Name filed in this class.
- ***Max*** (Long, required) – The maximum integer to be appended to the virtual identifier.
- ***Min*** (Long, required) – The minimum integer to be appended to the virtual identifier.
- ***StartingVirtualIndex*** (Long, required) – The first integer in a range of integers to be appended to the MapTo name.

3. Instantiation and execution of the IVI Configuration Servers

3.1 Installing the Configuration Server

The Configuration Server files are installed before any IVI software modules are installed. This is enforced by the requirements for installing IVI shared components, IVI instrument drivers and IVI-MSS role control modules described in *IVI-3.17: Installation Requirements Specification*.

The IVI-C and IVI-COM Configuration Server executable files are provided as both 32-bit and 64-bit Windows executables. The native IVI.NET assembly is an AnyCPU assembly. All executables, as well as associated data and schema files, are distributed with the IVI Shared Components.

3.1.1 Packaging

The IVI Configuration Server installation consists of the following files.

- IviConfigServer.dll – The COM API executable file.
- README.txt – A file that contains misc. information about the Configuration Server installation, including standard software dependencies, additional included files, and last minute instructions.
- IviConfigurationStore.xml – An empty XML configuration store file.
- IviConfigurationStore_1-6.xsd – The XSL schema for the configuration store file.
- Ivi.ConfigServer.Interop.dll – The .NET Primary Interop Assembly for the IVI-COM executable.
- Ivi.ConfigServer.Interop.xml – The .NET Primary Interop Assembly help file.
- IviConfigServerCapi.dll – The C API executable file
- IviConfigServer.lib – Microsoft C compatible library files.
- IviConfigServer.h – C header file for C API.
- Ivi.ConfigServer.dll – The native .NET API executable file.
- Ivi.ConfigServer.xml – The documentation file for the .NET Configuration Server.

The IVI Configuration Server installation for 64-bit Windows also includes 64-bit DLLs and libraries.

- IviConfigServer64.dll – The COM server executable file.
- IviConfigServerCapi64.dll – The C API executable file
- IviConfigServer64.lib – Microsoft C compatible library files.

The IVI Configuration Server C and COM implementations use Microsoft XML 6.0 (MSXML 6) if it is available on the target PC. If it is not available, the Configuration Server uses "Microsoft XML Core Services 4.0 RTM" which is installed when the IVI Configuration Server is installed. More information on these services is available at http://msdn.microsoft.com/downloads/default.asp?url=/downloads/sample.asp?url=/msdn-files/027/001/766/msdncompositedoc.xml.

The native .NET implementation uses the Microsoft .NET XmlReader and XmlWriter classes to deserialize and serialize config store files. These provide the best performance of the .NET XML I/O choices.

In addition, there may be other demo or test programs included – refer to the README.txt file for a complete listing of the files included in the installation.

3.1.2 Data File Installation

The installation directory for the master configuration store is <IVIDataDir>. The value of <IVIDataDir> is set by the IVI shared components installer. The master configuration store is installed as IviConfigurationStore.xml. The schema that describes the file format, IviConfigurationStore_1-6.xsd, is stored in the same directory as IviConfigurationStore.xml.

On 32-bit versions of Windows, the installation program creates the registry key `HKEY/LOCAL_MACHINE/SOFTWARE/IVI/CONFIGURATIONSERVER, MasterStore`, and sets the value to <IVIDataDir> \IviConfigurationStore.xml.

On Windows 7 (64-bit), Windows 8 (64-bit), and Windows 10 (64-bit), the installation program also creates the registry key HKEY/LOCAL_MACHINE/SOFTWARE/Wow6432Node/IVI/CONFIGURATIONSERVER, MasterStore, and sets the value to <IVIDataDir> \IviConfigurationStore.xml.

If the user renames or moves the master configuration store, the user must change the value of the registry key HKEY/LOCAL_MACHINE/SOFTWARE/IVI/CONFIGURATIONSERVER, MasterStore. On Windows 7 (64-bit), Windows 8 (64-bit), and Windows 10 (64-bit), the user must also change the value of the registry key HKEY/LOCAL_MACHINE/SOFTWARE/Wow6432Node/IVI/CONFIGURATIONSERVER, MasterStore to exactly the same value. To find the master configuration store, the Configuration Server looks at the location designated by the registry key. On Windows 7 (64-bit), Windows 8 (64-bit), and Windows 10 (64-bit), the Configuration Server looks at the location designated by both registry keys and returns an error if the values are not the same.

"The Configuration Server looks for the schema in <IVIDataDir>. For more information on the value of <IVIDataDir>, refer to Section 1.2, *Definition of Installation Terms*, of *IVI-3.17: Installer Requirements Specification."*

3.1.3 First Installation

When the Configuration Server is installed on a computer for the first time, there is only one entry in the master configuration store – the IviConfigStore entry. This entry carries component and version information, references collections of other Configuration Server objects.

The component and version information is read only, and serves to identify the Configuration Server component and version.

The collections referenced from the IviConfigStore object are empty. The collection methods and properties themselves may be executed, but will not return any configuration data until collection items (such as Software Modules, Hardware Assets, and so on) are added.

3.1.4 Subsequent Installations

When the Configuration Server is reinstalled on a system, it does not erase existing entries in the master configuration store. If the same version is being reinstalled, no changes will be made to the master configuration store. If a more recent version is being installed, there may be changes.

A re-installation may update the values of the IviConfigStore Revision, Description, Specification Major Version, and Specification Minor Version as appropriate.

A re-installation will also update or convert the data in the master configuration store to match any changes in the configuration store schema. If the re-installation cannot perform this task silently, a conversion utility will be provided by the IVI foundation as part of the new install, with suitable instructions.

Reasonable care will be taken to ensure that the Configuration Server behavior is compatible from revision to revision, and the IVI Foundation strongly recommends that all software access the master configuration store through the Configuration Server.

If either <IVIStandardDataDir> \IviConfigurationStore.xml or <IVIStandardDataDir> \IviConfigurationStore_1-6.xsd is missing, re-installing the Configuration Server shall create it as is done during an initial installation.

3.2 Accessing the Configuration Store

Users and software module developers should only access the configuration store through the Configuration Server software provided by the IVI Foundation. While it is possible to edit the configuration store XML file directly, it is discouraged because of the potential for introducing invalid data

and relationships into the file. Furthermore, the Configuration Server is designed to be independent of the way the data is stored, and insulates users from potential changes in the data format.

The Master Location property in the IviConfigStore class returns the full pathname of the master configuration store. It determines the name by reading the registry key `HKEY_LOCAL_MACHINE/SOFTWARE/IVI/CONFIGURATIONSERVER, MasterStore`, and, on Windows 7 (64-bit), Windows 8 (64-bit), and Windows 10 (64-bit), the registry key `HKEY_LOCAL_MACHINE/SOFTWARE/Wow6432Node/IVI/CONFIGURATIONSERVER, MasterStore`. Both values must match or an error is returned. For the precise semantics, refer to Section 7.3.3, *Master Location*.

Users may copy the master configuration store, modify it, and then save it to another file. Refer to Section 3.7, *Additional Instances of the Configuration Store*, for more details. Users may then designate such a file as the default copy of the configuration store for use in a process by assigning the full pathname of the file to the Process Default Location property in the IviConfigStore class. The Process Default Location property then returns the full pathname of this file. For the precise semantics, refer to Section 7.3.5, *Process Default Location*.

The IVI configuration store provides mechanisms for accessing the master and process default configuration store files.

Any utility, including Software Module installation programs, which modify the contents of the master configuration store should consider making a back up copy before serializing a modified version. The IVI Configuration Server tries to maintain the integrity of all configuration store files, but the consequences of a corrupt master configuration store are so severe that a back up could prove very valuable.

3.2.1 Master Configuration Store

To access the master configuration store using the C or COM servers
- Instantiate the IVI Configuration Server.
- Call the Deserialize method, providing the location of the master configuration store as a parameter. This is obtained through the Master Location property.
- Access the configuration data.

To access the master configuration store using the native .NET server
- Call the Load factory method without a parameter, which automatically loads the master configuration store. (Alternately, call the Load method with ConfigStoreLocation.Master, or with the full pathname of the master location as returned by the Master Location property.)
- Access the configuration data.

To modify the master configuration store using the C or COM servers
- Perform steps required to access the master configuration store
- Call the Serialize method with the master configuration store filename as a parameter.

To modify the master configuration store using the native .NET server
- Perform steps required to access the master configuration store
- Call the Save method with no parameter or with the master configuration store filename as a parameter.

When serializing to the master configuration store, care must be taken not to modify the data that is added when software modules are installed. For instance, deleting a software module entry could make it impossible to properly configure a driver session to access that software module.

3.2.2 Process Default Configuration Store

To set the process default configuration store for the current process
- Instantiate the IVI Configuration Server.

- Set the value of the Process Default Location property to the full path name of the configuration store file to be used in the current process.

To access the process default configuration store using the C or COM servers
- Set the process default configuration store for the current process (see previous step).
- Instantiate the IVI Configuration Server (if not already instantiated).
- Call the Deserialize method, providing the location of the process default configuration store as a parameter. This is obtained through the Process Default Location property.
- Access the configuration data.

To access the process default configuration store using the native .NET server
- Call the Load factory method with ConfigStoreLocation.ProcessDefault, or with the full pathname of the process default location as returned by the Process Default Location property.)
- Access the configuration data.

To modify the process default configuration store using the C or COM servers
- Perform steps required to access the process default configuration store.
- Call the Serialize method with the process default configuration store filename as a parameter.

To modify the process default configuration store using the native .NET server
- Perform steps required to access the master configuration store
- Call the Save method with the full pathname of the process default location as returned by the Process Default Location property.

3.2.3 Instantiating the Right Configuration Store from Software Modules

Software modules first try to open the process default configuration store. If the Process Default Location property is not the null string, and Deserialize (C/COM) or Load(.NET) cannot open the process default configuration store, it returns an error, and the software module returns an error as well. If the Process Default Location property is the null string then the software module tries to open the master configuration store. If Deserialize (C/COM) or Load(.NET) cannot open the master configuration store, it returns an error, and the software module returns an error as well.

3.2.4 Serializing to a Different Configuration Store

Users may wish to deserialize from one configuration store file and serialize to a different one. One basic example is copying a configuration store from one file to another. This is not part of the use model for driver installation or initialization.

3.3 Adding Entries to Collections

There are six “global” collections in the Configuration Server. Global collections include all of the instances of a particular class. The six global collections are
- Software Modules
- Published APIs
- Sessions
- Driver Sessions
- Hardware Assets
- Logical Names

When a new instance of one of the six associated classes is added using the Configuration Server, it is added first to the global collection. If a client tries to add it to another collection first, an error is returned. For example, if a software module adds a published API entry, it will add it to the global published APIs collection first, then add it to the Software Module’s published APIs collection.

3.4 Installing Software Modules

The configuration store must contain data describing a user-configurable software module before the user can configure the session or driver session that uses it. Installation requirements for software modules, including the configuration store entries, are described in *IVI-3.17: Installation Requirements Specification*.

A software module installation program creates the following types of entries in the master configuration store.

- A software module entry (required).
- A collection of references to published APIs (optional). If the published API entries do not exist, the software module installation program also adds the published API entry. The published API is added to the global published API collection first, and then added to the software module's published API collection.
- A collection of references to IVI Physical Names (optional).
- One or more collections of physical ranges, each associated with an IVI Physical Name (optional).
- A collection of references to data components, referenced by the software module (optional). Data components added when the software module is installed are called *module-defined data components*.
- A default session that uses the software module (optional).

The software module entry contains the information needed to create a running instance of the software module, as well as a couple of basic identification fields. This is required when IVI class drivers or the IVI Session Factory are used to instantiate an IVI specific driver.

The collection of IVI Physical Names identifies the repeated capabilities as defined in the software module. Refer to Section 2.9.2, *IVI Physical Name and IVI Physical Range* for more information.

3.4.1 Data Components In Software Modules

A software module's data components may be used for two basic purposes – to define initial settings for variables known to the software module, and to document the software module itself.

3.4.1.1 Defining Initial Settings

Software module data components may be used to document additional variables that are known by the software module, and that the software module will attempt to read from a configuration server session at run-time. For instance, the software module developer may define a Trace property that determines its tracing behavior, and decide that the software module will attempt to read an initial value for this variable from the configuration server. This variable is added to the data components of the software module when the software module is installed, to document the fact that the software module is capable of reading an initial value for it from the configuration server.

There are several reasons to add this type of data component to a software module.

- To provide additional configuration for driver operation. The IVI Foundation defines several properties for configuring driver operation (e.g. Cache, Simulate, InterchangeCheck, and so on), but a software module may need additional data. The Trace property mentioned above is an example of this type of data component.
- To provide additional configuration for instrument operation. In some cases, an instrument cannot be used interchangeably with others in its class because of some state variable that is not part of the class-compliant interfaces, but which must be set correctly in order for the class-compliant interfaces to work correctly. For example, an instrument that by default returns measurements in terms of period when the class-compliant interface returns frequency could use a data component that allows the user to configure the instrument to measure frequency. Refer to Section 5.10.1.5, *Applying Configurable Initial Settings*, in *IVI-3.1: Driver Architecture Specification*.
- To provide initial instrument state. In general, the IVI Foundation recommends against using the configuration server to store and restore instrument state. While it is possible, it is very complex and often redundant with other instrument functionality.

These data components are added with UsedInSession = "Required" or "Optional" and ReadOnly = True. If UsedInSession = "Required", the data component is copied by the configuration server to any session

when the reference property from the session to the software module is set or changed. If UsedInSession = "Optional", the configuration server will allow the data component to be added to the session.

3.4.1.2 Documenting the Software Module

Data components may be added to the software module just to add information about the software module. The developer may choose to add these when the software module is installed. These data components are added with UsedInSession = "None" to indicate that that they are not to be copied to a session for configuration.

3.4.2 Un-installing Software Modules

When deleting software modules,

- Delete collections for published APIs. Do not delete the published API entries.
- Delete collections and entries for physical names and ranges.
- Delete collections and entries for data components.
- Delete the software module entry.

To accomplish the above, use the Configuration Server to delete the software module entry. The Configuration Server will delete all of the above listed entries correctly.

When software module entries are deleted, the sessions that reference them will not be automatically deleted. These sessions may be reusable at a later time, after the software module is installed again, or they may be reusable with another compatible software module.

3.4.3 Re-installing Software Modules

When re-installing the same version of a software module, delete and re-add the following entries.

- The software module entry.
- Collections of published APIs. Do not delete the published API entries.
- Collections and entries for physical names and ranges.
- Collections and entries for data components.

To accomplish the above, use the Configuration Server to delete the software module entry. The Configuration Server will delete all of the above listed entries correctly. Then the installation will add the correct entries as part of the re-install process.

Re-installing is not a special feature. It may be implemented with an un-install followed by a normal install.

If the default session already exists, do not delete and re-add it. If it does not exist, add it.

When re-installing a different version of the software module, the above actions are taken. If the data components associated with the software module have been changed, the installer notifies the user that the older associated sessions are not compatible. (Note: what about silent installs? – log files, etc.)

3.5 Maintaining Configuration Data

Users add configuration data to the configuration store. Users have several mechanisms available for maintaining configuration data, including using a proprietary configuration store editor and using the Configuration Server from user application code. Users may manually edit the configuration store files, but this is strongly discouraged.

Users can configure several classes in the configuration store

- IVI Hardware Assets
- IVI Sessions and IVI Driver Sessions
- IVI Data Components associated with an IVI Session or an IVI Hardware Asset
- IVI Virtual Names and Ranges
- IVI Logical Names and Ranges

3.5.1 Configuring Hardware Assets

Configuration of hardware assets involves the following types of entries.

- Hardware asset entries. These must be added to the global hardware assets collection before being referenced by a session or driver session.
- A collection of references to data components, referenced by the hardware asset (optional). These collections and referenced data components are contained in the hardware asset.

Hardware asset entries identify the location of the instrument on the I/O buses. Users must add hardware asset entries for each instrument that is available for use by a session. In the future, vendors may provide instruments that make appropriate Hardware Asset entries.

There may be multiple entries with the same value for IO Resource Descriptor.

The hardware asset entries are not contained by the session. In the configuration store, they exist independently of the session, and are not deleted automatically when the session entry is deleted. A hardware asset may not be deleted if a session or driver session refers to it.

3.5.1.1 Data Components in Hardware Assets

Data components may be added to a hardware asset entry to further document the hardware asset. These data components are user-defined, since hardware assets are not added by software modules, with UsedInSession = "None". These may be added with the hardware asset, or at a later time. Refer to Section 3.5.5.1, IVI Hardware Assets for more details.

The hardware asset's data components collection is contained by the hardware asset that references it, and the associated data component entries are also contained by the hardware asset. They are added with and deleted with the hardware asset entry.

3.5.2 Configuring Sessions and Driver Sessions

Configuration of sessions and driver sessions involves the following types of entries.

- An IVI Session (required). The session entry will be a driver session entry if the referenced software module is an IVI instrument driver. These must be added to the global sessions collection and driver sessions collection (if applicable) before being referenced by a logical name.
- A collection of references to IVI Virtual Names (optional). These collections and referenced IVI Virtual Names are contained in the session.
- One or more collections of IVI Virtual Ranges, each associated with an IVI Virtual Name (optional). These collections and referenced physical names are contained in the session.
- A collection of references to IVI Data Components, referenced by the session (optional). These collections and referenced data components are contained in the session.

An IVI Session entry is used to configure a running instance of a software module. IVI instrument drivers are configured using IVI Driver Session, which inherit from IVI Session. Because of the inheritance, driver sessions include all of the functionality associated with sessions, and in addition allow the configuration of seven additional properties that have special meaning to IVI instrument drivers.

A session references an IVI Software Module. IVI class drivers and the IVI Session Factory both start with a reference to a session and examine the associated session to determine which software module to instantiate as described in Section 3.6.1, *IVI Class Drivers and the IVI Session Factory*. The information in the software module entry is sufficient to instantiate the software module – ProgID for IVI-COM, ModulePath for IVI-C. Both may be filled in if the same vendor provides both forms of the driver in the same installation, and one is a wrapper for the other.

IVI Sessions may reference zero or one IVI Hardware Asset, but this may not be enough for some software modules that require more than one hardware reference. There are two possible ways to handle this situation. First, IO Resource Descriptor may be overloaded by using a syntax that allows multiple instrument locations to be entered. Second, data components for the additional addresses may be added to the software module entry, and carried over to the session where the values are configured.

The session's IVI Virtual Name, IVI Virtual Range, and IVI Data Components collections are contained by the IVI Session that references them, and the associated virtual name, range, and data component entries are also contained by the session. They are added with and deleted with the session entry.

3.5.2.1 Virtual Names

3.5.3 The collection of IVI Virtual Names identifies the repeated capabilities as defined in the client, and maps these names to physical identifiers that are recognized by the software module. Refer to Section 2.9.2.3, *Uniqueness of IVI Physical Names* for more information.Data Components In Sessions

A session's data components may be used for two basic purposes – to configure initial settings for additional variables known to the associated software module, and to document the session itself.

3.5.3.1 Configurable Initial Settings

Software modules may use data components to allow configuration of software module variables at initialization. Refer to section 3.4.1.1, *Defining Initial Settings* for details. During initialization, the software module looks for these variables in the associated session that contains the configuration information. In order for this to work, the configuration server determines what additional variables are required when the reference to the session's Software Module is set by examining the data components for the software module, and copies all data components with UsedInSession = "Required" to the session. When it does the copy, it changes ReadOnly to "False". After the copy, clients may change the values of the session's data components to the correct values, and may add data components from the associated software module where UsedInSession = "Optional".

3.5.3.2 Documenting the Session

Data components may be added to the session just to add information about the session. The developer may choose to add these when the session is created, or they may be added by a configuration utility, test system, or configuration server user. These data components are added with UsedInSession = "None" to indicate that that they are not copied from the software module.

3.5.4 Configuring Logical Names

Configuration of logical identifiers involves the following types of entries.

- IVI Logical Name entries that reference session entries.

3.5.5 Documentation Data Components

Users or configuration utilities may add user-defined data components to hardware assets and sessions (including driver sessions). These are documentation data components, and there are no pre-defined uses for them – presumably the user or configuration utility that adds them knows why they are there.

3.5.5.1 IVI Hardware Assets

Users may add data components with Used In Session = "None", and Read Only = True or False. The data components are meaningful only to the particular user or configuration utility that added them.

3.5.5.2 IVI Sessions and IVI Driver Sessions

Users may add data components with Used In Session = "None", and ReadOnly = True or False. The data components are meaningful only to the particular user or configuration utility that added them.

3.6 Using Configuration Data

The Configuration Server is used to instantiate and initialize IVI instrument drivers and IVI-MSS role control modules. Instantiation is useful for class API interchangeability, using either IVI class drivers or the IVI Session Factory. As part of the Initialize (C/COM) or Load (.NET) functions, software modules read the configuration store and use the data to configure initial values.

3.6.1 IVI Class Drivers and the IVI Session Factory

IVI class drivers, the IVI-COM Session Factory, and IVI.NET Create session factory methods use the configuration store to identify and locate the IVI specific driver software module that they instantiate. The IVI Session Factory can be used to instantiate any kind of software module, including IVI-MSS role control modules. The user provides a logical name or a session or driver session name to the class driver, IVI Session Factory, or IVI.NET Create method. This name is then used to lookup the associated session or driver session entry, and the software module reference is then used to find the software module entry. The ModulePath or ProgID is then retrieved and used to instantiate the software module.

IVI class drivers, the IVI Session Factory, and the IVI.NET Create methods need only use Get Session to look up the session and then the software module. However, class drivers may choose to use Get Driver Session to be sure that the name passed in actually resolves to a driver session.

Get Session looks for the name first in the Configuration Server's logical name collection. If it finds it there, it follows the reference to the session. If it doesn't find the name in the logical name collection, it tries to find a session with the given name. If it doesn't find that, it returns an error. If Get Session finds the name in either place, it returns a pointer to the session.

Get Driver Session works the same as Get Session, except it is restricted to Driver Sessions.

3.6.2 Software Module Initialization

Once a software module is instantiated, it can use the Configuration Server as part of the initialization process.

An IVI instrument driver accesses the Configuration Server from the Initialize function. It queries the Configuration Server for the following information.

- Hardware Asset. The driver uses the IO Resource Descriptor to establish a connection to the instrument.
- The predefined driver properties - Cache, Driver Setup, Interchange Check, Query Instrument Status, Range Check, Record Coercions, and Simulate. These are applied in the Initialize function.
- Data components. A driver reads through the data components collection referenced by the session, looking for data component names that it recognizes. If it finds a data component that it doesn't recognize, it ignores it and continues with the next data component in the collection. After reading through the collection, if it hasn't retrieved the value of one or more required data components, it reports an error.
- Virtual identifiers. A driver reads through the virtual identifiers collection and any associated virtual ranges. It stores the mappings for use when resolving repeated capability names. Refer to section 2.9.3, *IVI Virtual Name and IVI Virtual Range* for more details.

For IVI-MSS role control modules, initialization happens whenever appropriate, as determined by the software module. Use of the hardware asset reference and virtual identifier, virtual range, and data component collections is analogous to the driver case. There are no pre-defined configuration variables for IVI-MSS that are analogous to the driver session properties.

3.6.3 Interchanging Instruments

Several sessions may be set up for a software module, representing several different ways of configuring the module. Since sessions are identified by name, just changing the name in the source code that

instantiates and configures the software module is enough to change the way it is configured. Different sessions can point to different hardware assets or different values for configuration properties.

In order to avoid any source code changes, use logical names to refer to sessions. If the logical name is used in the client's source code, the user can use a different session by changing the logical name's Session property.

3.7 Additional Instances of the Configuration Store

As mentioned above, it is possible to have multiple instances of the configuration store file. There is one master configuration store file on each system, but there may be additional configuration store files. The Process Default property can be set to the full pathname of any configuration store file. The Deserialize (C and COM) or Load (.NET) method may be called with any full pathname. Deserialize and Load will return an error if the file is not a legal Configuration Store file and cannot be successfully deserialized. Likewise, Serialize (C and COM) or Save (.NET) may be called with any full pathname, and will return an error if the file cannot be written.

Software module installers create entries only in the master configuration store. To create or maintain additional configuration store files, users must copy software module entries using the Configuration Server or use a configuration utility.

Users must be careful not to delete the master configuration store's software module entries, or to make modifications that would destroy the accuracy of the software module entries. This holds not only for the software module entry itself, but also for any associated data components, physical identifiers, or ranges.

3.8 Avoiding the Configuration Server

Users can avoid having to interact with the IVI Configuration Server and Store with most IVI specific instrument drivers and many other IVI software modules. In order to avoid interacting with the Configuration Server for IVI specific instrument drivers,

- The application program must explicitly specify the location of the IVI-C DLL, or the ProgID of the IVI-COM class.
- The Resource Name parameter of the Initialize function must be an IO resource descriptor, rather than a logical name or a session name.
- The application program must use the physical identifiers defined in the driver for repeated capabilities.
- Either the application program must use the software module's defaults for configurable attributes, or the application program must set the driver's attributes to initial values before the attributes are used in the driver. Note that adequate defaults are defined for all of the driver session properties in *IVI-3.2: Inherent Capabilities Specification*.

3.9 Copying Elements

In general, the IVI Configuration Store does not automatically copy data from one element to another. The user is responsible for getting the information from one element and then writing it into the second element.

The one exception happens when a Session's reference to a Software Module is set. Refer to Section 3.5.3.1, *Configurable Initial Settings*.

4. Collections

The configuration store design makes extensive use of collections of objects from a single class. For instance, the Configuration Store class includes a pointer to a collection of all of the IviHardwareAsset objects in the Configuration Store.

C and COM collections are one based. The smallest legal index is one which refers to the first item in the collection. .NET collections are dictionaries with string keys.

The IVI Configuration Server returns the "Not in Global Collection" error when a reference is made to an element before the element is added to the global collection. For example, a Software Module must be added to the Software Modules collection before the reference in a Session can be set to that Software Module.

The same element cannot be added to two collections in the same or different configuration stores. A second separate element with identical values can be used. The only exception is that a Published API can be added to the global Published APIs collection and to the Published APIs collection in a Software Module.

4.1 Collections in COM

In COM the designed collections are implemented as standard COM collections. Collection classes are indicated by appending "Collection" to the end of the class that describes the individual objects in the collection. For instance, if a collection points to objects of class IviHardwareAsset, the collection class would be named IviHardwareAssetCollection. The methods implemented by a COM collection are

- Item
- Count
- _NewEnum
- Add
- Remove

All of the above properties and methods have standard COM definitions. Refer to Microsoft documentation for more details.

Except for the Published API collection, the Item property and Remove method take one parameter whose type is VARIANT. The contents of the VARIANT may be either the name of an item in the collection or an integer which is the one-based index of an item in the collection. For the Published API collection, the parameters for Item and Remove are described in Section 7.2.4, Published APIs.

The _NewEnum property returns an IUnknown pointer. This interface can be queried for an IEnumVARIANT interface which contains the methods:

- Next
- Skip
- Reset
- Clone

The VARIANT returned by Next can be queried for an interface appropriate for the collection. For example the VARIANT returned within the Hardware Asset Collection can be queried for the IviHardwareAsset interface.

All of the above properties and methods have standard COM definitions. Refer to Microsoft documentation for more details.

The configuration server implements the following collection classes

- IviHardwareAssetCollection
- IviSessionCollection

- IviDriverSessionCollection
- IviPublishedAPICollection
- IviLogicalNameCollection
- IviSoftwareModuleCollection
- IviPhysicalNameCollection
- IviVirtualNameCollection
- IviPhysicalRangeCollection
- IviVirtualRangeCollection
- IviDataComponentCollection

4.2 Collections in C

In C, collections are accessed with generic functions defined for each class. These functions correspond to methods described for COM collections.

- IviConfig_Get<ItemName>Count
- IviConfig_Get<ItemName>ItemByIndex
- IviConfig_Get<ItemName>ItemByName
- IviConfig_Add<ItemName>Reference
- IviConfig_Remove<ItemName>Reference

where <ItemName> represents the name of the collection being accessed. For example, to get the number of IviSession objects in a given IviSessionsCollection, use the function IviConfig_GetSessionCount. To access a particular object in the collection, use the IviConfig_GetSessionItemByIndex function or the IviConfig_GetSessionItemByName function.

These functions appear only in the C API. They are equivalent to object creation and destruction handled by normal COM infrustructure.

- IviConfig_Create<ItemName>
- IviConfig_Destroy<ItemName>

To create a new item and add it to a collection, use the IviConfig_Create<ItemName> function defined for that collection. To remove an item from a collection, use the IviConfig_Destroy<ItemName> function defined for that collection. For example, to create a new IviSession object and add it to the global IviSessions collection, use the IviConfig_CreateSession function. To delete an IviSession object from the collection, use the IviConfig_DestroySession function. Note that objects can be created and deleted only from those collections that actually own the items.

To add a reference to an object to a collection that does not own the item, use the IviConfig_Add<ItemName>Reference function. To remove the reference, use the IviConfig_Remove<ItemName>Reference function.

4.3 Properties in C

In C, properties for the various objects are accessed using generic functions defined for each class. These function, which correspond to the get and put COM methods are:

- IviConfig_Get<ItemName>Property<DataType>
- IviConfig_Set<ItemName>Property<DataType>

where <ItemName> represents the name of the object for which the property is being accessed and <DataType> represents the data type of the property . For example, to get the value of the Revision property for an IviConfigStore object, call the IviConfig_GetConfigStorePropertyViString function.

All functions that return properties of type ViString comply with the rules in Section 3.1.2.1, *Additional Compliance Rules for C Functions with ViChar Array Output Parameters* of *IVI-3.2: Inherent Capabilities Specification*.

4.4 Return Codes

The IVI-3.2: Inherent Capabilities Specification defines general status codes that the collection functions can return.

The table below specifies additional IVI configuration server status codes for the Add function.

Completion Codes	Description
Not In Global Collection	The item does not exist in the global collection.
Duplicate Entry	An entry with name already exists.
Invalid Data Component	The data component is not a valid data component.

The table below specifies additional IVI configuration server status codes for the Item and Remove functions.

Completion Codes	Description
Does Not Exist	The item does not exist in the collection.

The table below specifies additional IVI configuration server status codes for the Remove function.

Completion Codes	Description
Reference Still Exists	The element cannot be removed from the global collection when it is referenced in the local collections.

The table below specifies additional IVI configuration server status codes for the Get and Set IVI-C functions.

Completion Codes	Description
Invalid Property ID	The specified property ID is not valid for this function.

4.5 Collections in .NET

Collection classes are indicated by appending “Collection” to the end of the class name that describes the individual objects in the collection. For instance, if a collection references objects of class HardwareAsset, the collection class would be named HardwareAssetCollection. The .NET configuration server implements the following collection classes:

- IviHardwareAssetCollection
- IviSessionCollection
- IviDriverSessionCollection
- IviPublishedAPICollection
- IviLogicalNameCollection
- IviSoftwareModuleCollection
- IviPhysicalNameCollection
- IviVirtualNameCollection

- IviPhysicalRangeCollection
- IviVirtualRangeCollection
- IviDataComponentCollection

The public methods exposed by a .NET collection are
- The index operator []
- Count
- Add
- Remove
- ContainsKey
- TryGetValue
- Clear
- GetEnumerator (per IEnumerable use patterns)

All of the above properties and methods have standard .NET definitions. Refer to Microsoft documentation for more details.

For most .NET collections, keys are specified with a single string parameter. The string is the name of an item in the collection. For the Published API collection, there are four parameters that identify the key value in methods and operators that take a key value. Refer to section 9.5, *IVI Published API Collection Methods and Operators (.NET Only)* for more details.

4.5.1 Collection Base Classes

.NET configuration server collections are derived from a set of base classes and interfaces. For the most part these base classes and interfaces are implementation details, but they are publically visible. This specification recommends that you always use the collection classes listed in the last section, and that you not use the base classes unless it is absolutely unavoidable. The base classes are:

- EntityCollection<TEntity> where TEntity : Entity
- EntityCollectionBase<TEntity> where TEntity : Entity
- IEntityCollection
- IEnumerable<T>

Refer to section 5.3.2, *.NET Entity Class*, for an overview of the Entity class.

5. C & .NET API Special Features

5.1 C API Special Features Overview

This section defines special functions for the IVI Configuration Server C API in addition to the functions defined for each IVI Configuration Server class. These functions are used to create an instance of the configuration server, dispose handles to IVI Configuration Server objects, and to retrieve and clear error codes and messages.

5.2 C API Special Functions

The IVI Configuration Server C API defines the following functions:

- Clear Error
- Close
- Dispose Handle
- Get Error
- Initialize

This section describes each function.

5.2.1 Clear Error

Description

This function clears the error description for the current thread of execution.

The Configuration Server C API logs its errors to the thread-local error variables defined by IVI-3.9 C Shared Component Specification. The Get Error function retrieves and clears these thread-local variables. For more information about thread local variables, refer to IVI-3.9 *C Shared Component Specification*, Section 7, *Thread Local Error Storage* for more information.

COM Method Prototype

```
N/A
```

C Function Prototype

```
ViStatus _VI_FUNC IviConfig_ClearError ();
```

Return Values

The *IVI-3.2: Inherent Capabilities Specification* defines general status codes that this function can return.

5.2.2 Close

Description

This function releases the handle to an IVI Configuration Store object.

Once a handle to the IVI Configuration Store object is no longer needed, the user must call this function to release the handle. Subsequent use of this handle will return the Invalid Handle error.

An application must release all IVI Configuration Store handles by calling this function before terminating. Failure to do so may result in resource or memory leaks.

COM Method Prototype

```
N/A
```

C Function Prototype

```
ViStatus _VI_FUNC IviConfig_Close (IviConfigStoreHandle ConfigStoreHandle);
```

Return Values

The *IVI-3.2: Inherent Capabilities Specification* defines general status codes that this function can return.

5.2.3 Dispose Handle

Description

This function releases the handle to an IVIConfigStore object returned from one of the Get Session, Get Driver Session, Get Collection, Create, Add Reference, Get Item, or Get Reference functions.

Once a handle to an item is no longer needed, the user must call this function to release the handle. Subsequent use of this handle will return the Invalid Handle error.

An application must release all handles by calling this function before terminating. Failure to do so may result in resource or memory leaks.

The user must not pass a handle of type IviConfigStoreHandle as the value of the Handle parameter. The user may pass a handle of any other type. If a handle of type IviConfigStoreHandle is passed as the value of the Handle parameter, this function will return the Invalid Handle error.

COM Method Prototype

N/A

C Function Prototype

```
ViStatus _VI_FUNC IviConfig_DisposeHandle (IviConfigHandle Handle);
```

Return Values

The IVI-3.2, *Inherent Capabilities Specification* defines general status codes that this function can return. In addition, it returns the following status codes:

Completion Codes	Description
Invalid Handle	The specified handle is invalid or of an incorrect type.

5.2.4 Get Error

Description

This function retrieves and clears the description of the first error that occurred for the current thread of execution.

One exception exists: If the `BufferSize` parameter is zero, the function does not clear the error description. By passing 0 for the buffer size, the caller can ascertain the buffer size required to get the entire error description string and then call the function again with a sufficiently large buffer.

The function complies with the rules in IVI-3.2, *Inherent Capabilities Specification,* Section 3.1.2.1, *Additional Compliance Rules for C Functions with ViChar Array Output Parameters*.

The Configuration Server C API logs its errors to the thread-local error variables defined by IVI-3.9 C Shared Component Specification. The Clear Error function clears these thread-local variables. For more information about thread local variables, refer to Section 7, Thread Local Error Storage in IVI-3.9 C Shared Component Specification for more information.

COM Method Prototype

`N/A`

C Function Prototype

```
ViStatus _VI_FUNC IviConfig_GetError (ViInt32 BufferSize,
                                      ViChar ErrorDescription[]);
```

Return Values

The *IVI-3.2: Inherent Capabilities Specification* defines general status codes that this function can return.

5.2.5 Initialize

Description

This function creates and returns a handle to a new instance of the IVI Configuration Store class. The user passes this handle to the C API functions defined for the IVI Configuration Store class to access its properties and to obtain handles to the IVI configuration store global collections.

This function may also perform additional initialization routines required for the use of the IVI Configuration Server C API. The user must first call this function before calling any other C API function. Every subsequent call to this function will create a new instance of the IVI Configuration Store class.

The user must call the Close function once and only once for each successful call to the Initialize function. Refer to Section 1.1.5 *Close* for more information.

COM Method Prototype

N/A

C Function Prototype

```
ViStatus _VI_FUNC IviConfig_Initialize (IviConfigStoreHandle*
                                        ConfigStoreHandle);
```

Return Values

The *IVI-3.2: Inherent Capabilities Specification* defines general status codes that this function can return.

5.3 .NET API Special Features Overview

This section defines special features of the .NET Configuration Server.

5.3.1 .NET Data Types

.NET uses the following basic data types in place of the corresponding COM types:

COM Data Type	.NET Data Type
BSTR	string
long	int
double	double
VARIANT_BOOL	bool

5.3.2 .NET Entity Class

Entity is an abstract base class that includes basic functionality used by other Config Store classes. Most of the class is not publically visible, but the class itself is visible and is used in several constructors including the main ConfigStore constructor.

The Entity class defines the following public property:

- Name

The following public Config Store classes are derived from Entity.

- HardwareAsset
- PublishedApi
- SoftwareModule
- PhysicalName
- PhysicalRange
- LogicalName
- Session
- DriverSession
- VirtualName
- VirtualRange
- DataComponent
- IviStructure
- IviInteger
- IviReal
- IviBoolean
- IviString
- IviAPIReference

For additional uses of the Entity class in .NET constructors, refer to 5.3.5.2, *Params Parameters in Constructors*.

5.3.2.1 The Entity.Name property.

The definition of Entity.Name matches the definition of the Name property in the classes that derive from entity.

In cases where a class places additional constriants on the Name property, the validation of Entity.Name is overridden by the class:

- Published Api
- IVI Physical Name
- IVI Virtual Name

5.3.3 .NET Enumerations

The .NET API defines the following enumerations:

- Config Store Location
- IVI Published API Name
- IVI Published API Type
- Session Usage

5.3.3.1 Config Store Location

The Config Store Location enumeration provides members for the two standard configuration store XML file locations.

Members do not have explicitly defined values.

Name	***Description***			
		Language	***Identifier***	***Value***
ConfigStoreLocation.Master	The master location.			
		.NET	`ConfigStoreLocation.Master`	
ConfigStoreLocation.ProcessDefault	The default location for the current process.			
		.NET	`ConfigStoreLocation.ProcessDefault`	

5.3.3.2 IVI Published API Name

The IVI Published API Name enumeration provides members for each of the standard APIs defined for IVI Instrument Drivers.

Members do not have explicitly defined values.

Name	***Description***			
		Language	***Identifier***	***Value***
IviPublishedApiName.IviDmm	The IviDmm instrument class.			
		.NET	`IviPublishedApiName.IviDmm`	
IviPublishedApiName.IviDriver	The IviDriver inherent capabilities class.			
		.NET	`IviPublishedApiName.IviDriver`	
IviPublishedApiName.IviScope	The IviScope instrument class.			
		.NET	`IviPublishedApiName.IviScope`	
IviPublishedApiName.IviFgen	The IviFgen instrument class.			
		.NET	`IviPublishedApiName.IviFgen`	
IviPublishedApiName.IviDCPwr	The IviDCPwr instrument class.			
		.NET	`IviPublishedApiName.IviDCPwr`	
IviPublishedApiName.IviACPwr	The IviACPwr instrument class.			
		.NET	`IviPublishedApiName.IviACPwr`	

IviPublishedApiName.IviSwtch	The IviSwtch instrument class.			
		.NET	`IviPublishedApiName.IviSwtch`	
IviPublishedApiName.IviPwrMeter	The IviPwrMeter instrument class.			
		.NET	`IviPublishedApiName.IviPwrMeter`	
IviPublishedApiName.IviSpecAn	The IviSpecAn instrument class.			
		.NET	`IviPublishedApiName.IviSpecAn`	
IviPublishedApiName.IviRFSigGen	The IviRFSigGen instrument class.			
		.NET	`IviPublishedApiName.IviRFSigGen`	
IviPublishedApiName.IviCounter	The IviCounter instrument class.			
		.NET	`IviPublishedApiName.IviCounter`	
IviPublishedApiName.IviDownConverter	The IviDownconverter instrument class.			
		.NET	`IviPublishedApiName.IviDownconverter`	
IviPublishedApiName.IviUpConverter	The IviUpconverter instrument class.			
		.NET	`IviPublishedApiName.IviUpconverter`	
IviPublishedApiName.IviDigitizer	The IviDigitizer instrument class.			
		.NET	`IviPublishedApiName.IviDigitizer`	
IviPublishedApiName.IviLxiSync	The IviLxiSync instrument class.			
		.NET	`IviPublishedApiName.IviLxiSync`	

5.3.3.3 IVI Published API Type

The IVI Published API Type enumeration provides members for the three standard API types defined for IVI Instrument Drivers.

Members do not have explicitly defined values.

Name	***Description***			
		Language	***Identifier***	***Value***
IviPublishedApiType.IviCom	The IVI-COM API type.			
		.NET	`IviPublishedApiType.IviCom`	
IviPublishedApiType.IviNet	The IVI.NET API type.			
		.NET	`IviPublishedApiType.IviNet`	
IviPublishedApiType.IviC	The IVI-C API type.			
		.NET	`IviPublishedApiType.IviC`	

5.3.3.4 Session Usage

The Config Store Location enumeration provides members for the three standard values for data component Used In Session properties.

Name	*Description*			
		Language	*Identifier*	*Value*
SessionUsage.None	A session does not need to define a value for this data component.			
		.NET	`SessionUsage.None`	`0`
SessionUsage. Required	A session must define a value for this data component.			
		.NET	`SessionUsage.Required`	`1`
SessionUsage. Optional	A session may define a value for this data component.			
		.NET	`SessionUsage.Optional`	`2`

5.3.4 Collections in .NET

Refer to section 4.5, *Collections in .NET* for a description of collections in .NET.

5.3.5 .NET Constructors

In contrast to the COM API which does not allow parameterized constructors, the .NET API uses parameterized constructors, so that new objects have legal and consistent data from the start.

5.3.5.1 Constructors and Property Access

Each constructor includes parameters for

- Properties that serve as keys in the corresponding collection. For example, Name is the key value for nearly all classes, and so name is a common constructor parameter.
- Properties that should not be changed after they are set initially. For example, the Prefix property for SoftwareModules only needs to be set once, and does not need to be changed after that.

In general, if a constructor includes a parameter, the corresponding property is read only.

5.3.5.2 Params Parameters in Constructors

Some constructors include a params parameter. In the .NET API, these allow a calling program to specify members of the object's collections. For example, the constructor for the Virtual Name class includes the `virtualRanges` parameter, which allows the calling program to specify multiple Virtual Range objects to be added to the Virtual Name's Virtual Range collection.

In most cases the type of the params parameter is an array of a base type, either Entity or DataComponent. This allows the calling program to specify any object whose class derives from the base class. The constructor is then responsible for ensuring that the specified objects are valid for that constructor. For example, the Config Store constructor has a params parameter of type Entity[]. This allows the calling program to specify any class that derives from Entity as an argument. However, the Config Store constructor checks each argument to make sure that they are valid for the constructor, and only six types are valid: Driver Sessions, Hardware Assets, Logical Names, Published APIs, Sessions, and Software Modules. If the calling program specifies an object of an invalid derived type, the constructor will throw an exception.

Constructors may enforce additional order constraints on the specified objects. The most common constraint is for items that must be added to one of the six global collections before they are referenced elsewhere. If an order constraint is violated, the constructor will throw an exception.

5.3.5.3 Collection Constructors

Collection constructors are not public. Instead, the Configuration Server creates collections when needed, and allows calling programs to populate the collections using the collection Add() methods or constructor params arguments.

5.3.6 .NET Static Methods and Properties

The .NET API includes a few static methods and properties. These are documented for the classes where they are defined, in separate sections to emphasize the fact that they are static. For the most part these methods and properties do not have an exact match in the C or COM APIs.

The most important of the static methods is the factory method Load(…). Load combines instantiating a ConfigStore object and deserializing a configuration store file, returning the instantiated and loaded object. In contract, the ConfigStore constructor should only be used to create a new configuration store from scratch.

5.3.7 .NET Schema Validation

The .NET API provides more control over validation than the C or COM APIs. The default for all of the APIs is to validate an XML file against the Configuration Store XML schema when a file is deserialized. In addition, the .NET API allows a calling program to deserialize a file without validation (for performance) and to validate a file without deserializing it.

5.3.8 .NET Exceptions

Refer to section 5.12.2, *IVI.NET Error Handling*, of *IVI-3.1, Driver Architecture Specification*, for a general overview of exceptions in IVI.NET components. Calling programs should be able to accommodate arbitrary exceptions, including those defined by the .NET Framework.

The .NET Configuration Server does not define any new exceptions. When it explicitly throws an exception, it throws one of the standard .NET exceptions (usually ArgumentException or InvalidOperationException) with a custom message that indicates the problem.

The Configuration Server does not generate warnings.

6. IVI Configurable Components Class (Virtual)

6.1 IVI Configurable Components Overview

The IVI Configurable Components class allows developers to add properties to several of the other Configuration Server classes. This class is not implemented directly – it is a virtual base class. The following Configuration Server classes inherit from the IVI Configurable Components class:

- IVI Session
- IVI Driver Session (through IVI Session)
- IVI Hardware Asset
- IVI Software Module

Each of these classes inherits Name, Description, and a reference to a collection of Data Component objects. Each object in the Data Component collection represents either a property added by the developer or a pointer to another collection of Data Component objects.

6.2 IVI Configurable Components References

The IVI Configurable Components class defines the following reference:

- Data Components

This section describes the reference.

6.2.1 Data Components

API Technology	Data Type	Access
COM	IIviDataComponentCollection**	R/O
C	IviDataComponentCollectionHandle	R/O
.NET	DataComponentCollection	R/O

COM/.NET Property Name

DataComponents

C Function Prototype

```
ViStatus _VI_FUNC IviConfig_GetConfigComponentDataComponentCollection
                                                (IviConfigComponentHandle
                                                ConfigComponentHandle,
                                                IviDataComponentCollectionHandle*
                                                DataComponentCollectionHandle);
```

C Parameters

Inputs	Description	Datatype
ConfigComponentHandle	Handle to an IviConfigComponent object. You may pass a handle to any of the derived IviConfigComponent objects.	IviConfigComponentHandle

Outputs	Description	Datatype
DataComponentCollectionHandle	Handle to an IviDataComponentCollection object.	IviDataComponentCollectionHandle

Description

References a collection of DataComponents that modifies the object of which the collection is a part.

6.3 IVI Configurable Components Properties

The IVI Configurable Components class defines the following properties:

- Description
- Name

This section describes the behavior and requirements of each property.

6.3.1 Description

API Technology	Data Type	Access
.NET	string	R/W
C	ViString	R/W
COM	BSTR	R/W

COM/.NET Property Name

Description

C Constant Name

IVICONFIG_VAL_CONFIG_COMPONENT_DESCRIPTION

Description

The description of the associated object. The empty string is a legal value for this property.

6.3.2 Name

API Technology	Data Type	Access
.NET	string	R/O
C	ViString	R/W
COM	BSTR	R/W

COM/.NET Property Name

Name

C Constant Name

IVICONFIG_VAL_CONFIG_COMPONENT_NAME

Description

The name of the associated object. The empty string is not a legal value for this property.

7. IVI Configuration Store Class

7.1 IVI Configuration Store Overview

The IVI Configuration Store class is the main class of the Configuration Server. There is exactly one IVI Configuration Store object in each instance of the configuration server. This object is created before any others, and there is no way to delete it. Use of the Configuration Server starts with this object.

The IVI Configuration Store class allows users to find out information about the Configuration Server using a similar approach to that used in other IVI components. Information includes Revision, Specification Major Version, Specification Minor Version, and Vendor, as well as Name and Description.

The IVI Configuration Store class allows users to Deserialize (Load) an IVI configuration store XML file into the Configuration Server, and to Serialize (Save) updated information out to the file again.

The IVI Configuration Store class provides two helper functions to help developers find Sessions and Driver Sessions in the configuration store. Sessions and Driver Sessions may be identified either by their Name or by a Logical Name that maps to their name. These functions are be used to make sure that the logic used to search for a session is correct and consistent among IVI software modules.

Finally, the IVI Configuration Store class provides the means to navigate to collections of Configuration Store objects. This class includes references to collections of Logical Names, Sessions, Driver Sessions, Hardware Assets, Software Modules, and Published APIs.

7.2 IVI Configuration Store References

The IVI Configuration Store class defines the following references:

- Driver Sessions
- Hardware Assets
- Logical Names
- Published APIs
- Sessions
- Software Modules

This section describes each reference.

7.2.1 Driver Sessions

API Technology	Data Type	Access
COM	IIviDriverSessionCollection**	R/O
C	IviDriverSessionCollectionHandle	R/O
.NET	DriverSessionCollection	R/O

COM/.NET Property Name

DriverSessions

C Function Prototype

```
ViStatus _VI_FUNC IviConfig_GetConfigStoreDriverSessionCollection
                                                (IviConfigStoreHandle
                                                ConfigStoreHandle,
                                                IviDriverSessionCollectionHandle*
                                                DriverSessionCollectionHandle);
```

C Parameters

Inputs	Description	Datatype
ConfigStoreHandle	Handle to an IviConfigStore object.	IviConfigStoreHandle

Outputs	Description	Datatype
DriverSessionCollectionHandle	Handle to an IviDriverSessionCollection object.	IviDriverSessionCollectionHandle

Description

References the global collection of all Driver Session objects in the configuration store.

7.2.2 Hardware Assets

API Technology	Data Type	Access
COM	IIviHardwareAssetCollection**	R/O
C	IviHardwareAssetCollectionHandle	R/O
.NET	HardwareAssetCollection	R/O

COM/.NET Property Name

HardwareAssets

C Function Prototype

```
ViStatus _VI_FUNC IviConfig_GetConfigStoreHardwareAssetCollection
                                                (IviConfigStoreHandle
                                                ConfigStoreHandle,
                                                IviHardwareAssetCollectionHandle*
                                                HardwareAssetCollection);
```

C Parameters

Inputs	Description	Datatype
ConfigStoreHandle	Handle to an IviConfigStore object.	IviConfigStoreHandle

Outputs	Description	Datatype
HardwareAssetCollection	Handle to an IviHardwareAssetCollection object.	IviHardwareAssetCollectionHandle

Description

References the global collection of all Hardware Asset objects in the configuration store.

7.2.3 Logical Names

API Technology	Data Type	Access
COM	IIviLogicalNameCollection**	R/O
C	IviLogicalNameCollectionHandle	R/O
.NET	LogicalNameCollection	R/O

COM/.NET Property Name

LogicalNames

C Function Prototype

```
ViStatus _VI_FUNC IviConfig_GetConfigStoreLogicalNameCollection
                                                (IviConfigStoreHandle
                                                ConfigStoreHandle,
                                                IviLogicalNameCollectionHandle*
                                                LogicalNameCollectionHandle);
```

C Parameters

Inputs	Description	Datatype
ConfigStoreHandle	Handle to an IviConfigStore object.	IviConfigStoreHandle

Outputs	Description	Datatype
LogicalNameCollectionHandle	Handle to an IviLogicalNameCollection object.	IviLogicalNameCollectionHandle

Description

References the global collection of all Logical Name objects in the configuration store.

7.2.4 Published APIs

API Technology	Data Type	Access
COM	IIviPublishedAPICollection**	R/O
C	IviPublishedAPICollectionHandle	R/O
.NET	PublishedAPICollection	R/O

COM/.NET Property Name

PublishedAPIs

C Function Prototype

```
ViStatus _VI_FUNC IviConfig_GetConfigStorePublishedAPICollection
                                                    (IviConfigStoreHandle
                                                    ConfigStoreHandle,
                                                    IviPublishedAPICollectionHandle*
                                                    PublishedAPICollection);
```

C Parameters

Inputs	Description	Datatype
ConfigStoreHandle	Handle to an IviConfigStore object.	IviConfigStoreHandle

Outputs	Description	Datatype
PublishedAPICollection	Handle to an IviPublishedAPICollection object.	IviPublishedAPICollectionHandle

Description

References the global collection of all Published API objects in the configuration store.

The Item property and Remove method for this collection require parameters different from the other collections.

```
Item( [in] VARIANT varIndex, [in] long MajorVersion, [in] long MinorVersion,
      [in] BSTR Type, [out, retval]IIviPublishedAPI**pVal)
```

```
Remove( [in] VARIANT varIndex, [in] long MajorVersion, [in] long MinorVersion,
        [in] BSTR Type, [out, retval]IIviPublishedAPI**pVal)
```

If the parameter, varIndex, is an integer, the remaining parameters are ignored as a numeric index completely identifies the item to be retrieved or removed.

The key for the Published API Collection is the combination of the PublishedAPI Name, Type, Major Version, and Minor Version. This is the only collection in the Configuration Server that has a composite key. Collection methods such as Item and Remove that require a key have a set of four parameters that comprise the key.

7.2.5 Sessions

API Technology	Data Type	Access
COM	IIviSessionCollection**	R/O
C	IviSessionCollectionHandle	R/O
.NET	SessionCollection	R/O

COM/.NET Property Name

Sessions

C Function Prototype

```
ViStatus _VI_FUNC IviConfig_GetConfigStoreSessionCollection
                                                (IviConfigStoreHandle
                                                ConfigStoreHandle,
                                                IviSessionCollectionHandle*
                                                SessionCollectionHandle);
```

C Parameters

Inputs	Description	Datatype
ConfigStoreHandle	Handle to an IviConfigStore object.	IviConfigStoreHandle

Outputs	Description	Datatype
SessionCollectionHandle	Handle to an IviSessionCollection object.	IviSessionCollectionHandle

Description

References the global collection of all IVI Session objects in the configuration store. The collection of all sessions includes all driver sessions.

7.2.6 Software Modules

API Technology	Data Type	Access
COM	IIviSoftwareModuleCollection**	R/O
C	IviSoftwareModuleCollectionHandle	R/O
.NET	SoftwareModuleCollection	R/O

COM/.NET Property Name

SoftwareModules

C Function Prototype

```
ViStatus _VI_FUNC IviConfig_GetConfigStoreSoftwareModuleCollection
                                                (IviConfigStoreHandle
                                                ConfigStoreHandle,
                                                IviSoftwareModuleCollectionHandle*
                                                SoftwareModuleCollection);
```

C Parameters

Inputs	Description	Datatype
ConfigStoreHandle	Handle to an IviConfigStore object.	IviConfigStoreHandle

Outputs	Description	Datatype
SoftwareModuleCollection	Handle to an IviSoftwareModuleCollection object.	IviSoftwareModuleCollectionHandle

Description

References the global collection of all Software Module objects in the configuration store.

7.3 IVI Configuration Store Properties

The IVI Configuration Store class defines the following properties:

- Actual Location
- Description
- Master Location
- Name
- Process Default Location
- Revision
- Specification Major Version
- Specification Minor Version
- Vendor

This section describes the behavior and requirements of each property.

7.3.1 Actual Location

API Technology	Data Type	Access
.NET	string	R/O
C	ViString	R/O
COM	BSTR	R/O

COM/.NET Property Name

ActualLocation

C Constant Name

IVICONFIG_VAL_CONFIG_STORE_ACTUAL_LOCATION

Description

Returns the full pathname of the configuration store file that is currently being edited. In general, this value is set when the configuration server deserializes or serializes a configuration store file. If no configuration store file has been successfully deserialized by the current instance of the configuration server, the property returns the empty string.

7.3.2 Description

API Technology	Data Type	Access
.NET	string	R/O
C	ViString	R/W
COM	BSTR	R/W

COM/.NET Property Name

Description

C Constant Name

IVICONFIG_VAL_CONFIG_STORE_DESCRIPTION

Description

The description of the Configuration Server component. The default value for this string is “The IVI Configuration Server allows access to and modification of an IVI configuration store.”

7.3.3 Master Location

API Technology	Data Type	Access
.NET	string	R/O
C	ViString	R/O
COM	BSTR	R/O

COM/.NET Property Name

MasterLocation

C Constant Name

IVICONFIG_VAL_CONFIG_STORE_MASTER_LOCATION

Description

Specifies the full pathname of the master configuration store. This includes the file name, which is always IviConfigurationStore.xml.

The configuration server checks the Windows registry for a non-empty value of the following registry key:
HKEY_LOCAL_MACHINE\SOFTWARE\IVI\CONFIGURATIONSERVER, MasterStore
If MasterStore exists, the value of Master Location shall be the value of this key. If MasterStore does not exist or is an empty-value, the configuration server shall return Master Not Found error.

On 64-bit operating systems, the configuration server also checks the Windows registry for a non-empty value of the following registry key:
HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\IVI\CONFIGURATIONSERVER, MasterStore
If MasterStore does not exist, the configuration server shall return Master Not Found error.

On 64-bit operating systems, the configuration server compares the values of the above two MasterStore locations. If both registry keys are not set to the same value, the configuration server shall return the Master Store Registry Conflict error.

Return Values

The *IVI-3.2: Inherent Capabilities Specification* defines general status codes that this function can return.

The table below specifies additional IVI configuration server status codes for this function.

Completion Codes	Description
Master Not Found	The registry key does not exist or the file can not be found.
Master Store Registry Conflict	The locations of the master configuration store in the 32-bit and 64-bit registry hives are not the same.

7.3.4 Name

API Technology	Data Type	Access
.NET	string	R/O
C	ViString	R/O
COM	BSTR	R/O

COM/.NET Property Name

Name

C Constant Name

IVICONFIG_VAL_CONFIG_STORE_NAME

Description

The name of the Configuration Server executable.

- For a 32-bit COM executable, “IVI Configuration Server (32-bit COM)”
- For a 64-bit COM executable, “IVI Configuration Server (64-bit COM)”
- For a 32-bit C executable, “IVI Configuration Server (32-bit C)”
- For a 64-bit C executable, “IVI Configuration Server (64-bit C)”

7.3.5 Process Default Location

API Technology	Data Type	Access
.NET	string	R/W
C	ViString	R/W
COM	BSTR	R/W

COM/.NET Property Name

ProcessDefaultLocation

C Constant Name

IVICONFIG_VAL_CONFIG_STORE_PROC_DEFAULT_LOCATION

Description

Specifies the full pathname of the default configuration store for the process in which this property is used. If a non-empty string is assigned to this property by code executed in the process, the property shall return that string when accessed in the same process. The empty string is a legal value for this property. Refer to Section 0, *To modify the process default configuration store using* the native .NET server

- Perform steps required to access the master configuration store
- Call the Save method with the full pathname of the process default location as returned by the Process Default Location property.

Instantiating the Right Configuration Store from Software Modules, for more information.

Implementation Note

Once the value of Process Default Location is determined by the first instance of the Configuration Server in a process, the value is stored in an environment variable called “IVICONFIGSERVERDEFAULT”. Subsequent instances of the Configuration Server in the same process retrieve the value of this environment variable and return it as the value of the Process Default Location property”

7.3.6 Revision

API Technology	Data Type	Access
.NET	string	R/O
C	ViString	R/O
COM	BSTR	R/O

COM/.NET Property Name

Revision

C Constant Name

IVICONFIG_VAL_CONFIG_STORE_REVISION

Description

This string shall be the current revision of the Configuration Server. The format of the revision string or a revision string is defined in Section 5.18, *File Versioning*, of the *IVI-3.1: Driver Architecture Specification*.

7.3.7 Specification Major Version

API Technology	Data Type	Access
.NET	int	R/O
C	ViInt32	R/O
COM	long	R/O

COM/.NET Property Name

SpecificationMajorVersion

C Constant Name

IVICONFIG_VAL_CONFIG_STORE_SPEC_MAJOR_VERSION

Description

This property shall be the major version of the Configuration Server specification supported by the Configuration Server component. The rules related to Specification Major Version are defined in Section 5.18, *File Versioning*, of the *IVI-3.1: Driver Architecture Specification*.

7.3.8 Specification Minor Version

API Technology	Data Type	Access
.NET	int	R/O
C	ViInt32	R/O
COM	long	R/O

COM/.NET Property Name

SpecificationMinorVersion

C Constant Name

IVICONFIG_VAL_CONFIG_STORE_SPEC_MINOR_VERSION

Description

This property shall be the minor version of the Configuration Server specification supported by the Configuration Server component. The rules related to Specification Minor Version are defined in Section 5.18, *File Versioning*, of the *IVI-3.1: Driver Architecture Specification*.

7.3.9 Vendor

API Technology	Data Type	Access
.NET	string	R/O
C	ViString	R/O
COM	BSTR	R/O

COM/.NET Property Name

Vendor

C Constant Name

IVICONFIG_VAL_CONFIG_STORE_VENDOR

Description

The vendor of the Configuration Server component. This string shall be “IVI Foundation, Inc.”.

7.4 IVI Configuration Store Functions

The IVI Configuration Store class defines the following functions:

- Deserialize
- Get Driver Session
- Get Session
- Serialize

This section describes the behavior and requirements of each function.

7.4.1 Deserialize (C and COM Only)

Description

Reads a configuration store file from a data source location, parses the data, and creates the corresponding Configuration Server classes in memory.

Deserialize opens the configuration file specified by the Location parameter. The location parameter must be the full pathname of the configuration store file to be opened. If the file is found, but cannot be successfully deserialized, Deseriailize shall return a Deserialized Failed error and the Configuration Server is returned to its initial state.

Deserialize may only be run successfully once per instance of the Configuration Server. If Deserialize is called after being previously called successfully it shall return a Already Deserialized error. Multiple copies of the Configuration Server can be accessed by accessing multiple instances of the Configuration Server.

COM Method Prototype

```
Deserialize([in] BSTR Location);
```

C Function Prototype

```
ViStatus _VI_FUNC IviConfig_Deserialize (IviConfigStoreHandle
                                ConfigStoreHandle, ViConstString Location);
```

Return Values

The *IVI-3.2: Inherent Capabilities Specification* defines general status codes that this function can return.

The table below specifies additional IVI configuration server status codes for this function.

Completion Codes	Description
Deserialize Failed	The specified Configuration Store file could not be deserialized.
Already Deserialized	A deserialize was attempted after a previous de-serialize had already succeeded.

7.4.2 Get Driver Session

Description

Returns a reference to a driver session, given a name that identifies the session. Name may be either the Name of the Driver Session object, or a logical name that refers to the Driver Session object.

To find the Driver Session object, Get Driver Session first searches the global Logical Names collection to see if Name is defined in that collection. If Name is found in the collection, Get Driver Session examines the referenced Session. If the Session is a Driver Session, then the Driver Session reference is returned in the DriverSession parameter.

If Name is not found in the Logical Names collection, or if the Session referenced by the Logical Name is not a Driver Session, Get Driver Session searches the global Driver Session collection to see if Name is defined in the Driver Session collection. If Name is found in the Driver Session collection, then the Driver Session reference is returned in the DriverSession parameter.

If Name is not found in the Logical Names or the Driver Session collections, Get Driver Session shall return a NULL pointer for the DriverSession parameter and shall return a Session Not Found error. Note that if both collections have an item that matches Name, the item found by following the Logical Name reference is returned.

COM Method Prototype

```
GetDriverSession([in] BSTR Name,
                 [in,out] IIviDriverSession** DriverSession);
```

.NET Method Prototype

```
public DriverSession GetDriverSession(string name)
```

C Function Prototype

```
ViStatus _VI_FUNC IviConfig_GetDriverSession (IviConfigStoreHandle
                                    ConfigStoreHandle,
                                    ViConstString Name,
                                    IviDriverSessionHandle*
                                    DriverSessionHandle);
```

Return Values

The *IVI-3.2: Inherent Capabilities Specification* defines general status codes that this function can return.

The table below specifies additional IVI configuration server status codes for this function.

Completion Codes	Description
Session Not Found	The session name or logical name could not be resolved to a driver session.

7.4.3 Get Session

Description

Returns a reference to a session, given a name that identifies the session. Name may be either the Name of the Session object, or a logical name that refers to the Session object.

To find the Session object, Get Session first searches the global Logical Names collection to see if Name is defined in that collection. If Name is found in the Logical Names collection, then the Session reference is returned in the Session parameter.

If Name is not found in the collection, Get Session searches the global Sessions collection to see if Name is defined in the Sessions collection. If Name is found in the collection, then the Session reference is returned in the Session parameter.

If Name is not found in the Logical Names or the Sessions collections, Get Session returns a NULL pointer for the Session parameter and returns a Session Not Found error.

Note that if both collections have an item that matches Name, the item found by following the Logical Name reference is returned.

COM Method Prototype

```
GetSession([in] BSTR Name,
           [in,out] IIviSession** Session);
```

.NET Method Prototype

```
public Session GetSession(string name)
```

C Function Prototype

```
ViStatus _VI_FUNC IviConfig_GetSession (IviConfigStoreHandle ConfigStoreHandle,
                                        ViConstString Name,
                                        IviSessionHandle* SessionHandle);
```

Return Values

The *IVI-3.2: Inherent Capabilities Specification* defines general status codes that this function can return.

The table below specifies additional IVI configuration server status codes for this function.

Completion Codes	Description
Session Not Found	The session name or logical name could not be resolved to a session.

7.4.4 Serialize (C and COM Only)

Description

Serializes a configuration store from the Configuration Server to a data source location. The Serialize method creates the configuration file specified by the Location parameter. The location parameter must be the full pathname of the configuration store file to be written, or a path relative to the current working directory. If the folders specified in the pathname do not exist, this function will create them. If the file is found, but cannot be successfully serialized, Serialize shall return a Serialize Failed error.

COM Method Prototype

```
Serialize([in] BSTR Location)
```

C Function Prototype

```
ViStatus _VI_FUNC IviConfig_Serialize (IviConfigStoreHandle ConfigStoreHandle,
                                       ViConstString Location);
```

Return Values

The *IVI-3.2: Inherent Capabilities Specification* defines general status codes that this function can return.

The table below specifies additional IVI configuration server status codes for this function.

Completion Codes	Description
Serialize Failed	The specified Configuration Store file could not be serialized.

7.4.5 Add (.NET Only)

Description

Adds a set of items to the six global collections in the configuration store.

Children parameters may be any one of the following classes deriving from Entity:

- PublishedApi,
- SoftwareModule
- HardwareAsset
- DriverSession
- Session
- LogicalName

Children parameters are added to the corresponding global collections in the order in which they are specified. Order is important because if a child parameter references another child parameter, the referenced item must have already been added to its corresponding global collection.

.NET Prototype

```
public void Add(params Entity[] children)
```

7.4.6 Save (.NET Only)

Description

Serializes a configuration store from the Configuration Server to a data source location. The Serialize method creates the configuration file specified by the Location parameter. The location parameter must be the full pathname of the configuration store file to be written, or a path relative to the current working directory. If the folders specified in the pathname do not exist, this function will create them.

.NET Prototype

```
public void Save (ConfigStoreLocation location = ConfigStoreLocation.Master)

public void Save (string location)
```

7.5 IVI Configuration Store Constructor (.NET Only)

The .NET IVI Configuration Store class defines one public constructor.

This section describes the behavior and requirements of the constructor.

7.5.1 ConfigStore Constructor

Description

Creates an instance of a configuration server.

If there are no children parameters, the ConfigStore is empty, and may be serialized to create a "default" configuration store XML file that is logically the equivalent of the file provided when the ConfigServer component is installed.

Children parameters may be any one of the classes in the list below. These classes derive from the Entity class. They are added in the order shown in the list below, which guarantees that referenced entities are added before the entities that reference them.

- PublishedApi,
- HardwareAsset
- SoftwareModule
- Session
- DriverSession
- LogicalName

In addition to using this constructor, new instances of the .NET configuration server may also be created using the following static factory methods:

- Load (refer to section 7.7.1, *Load*)
- Load Without Validation (refer to section 7.7.2, *Load Without Validation*)

.NET Prototype

```
public ConfigStore(params Entity[] children)
```

.NET Parameters

Inputs	Description	.NET Type
`children`	Published APIs, Software Modules, Hardware Assets, Driver Sessions, Sessions, and Logical Names to be added to the corresponding global collections referenced by the Config Store. There may be zero to n children of each type.	`params Entity[]`

7.6 IVI Configuration Store Static Property (.NET Only)

The .NET IVI Configuration Store class defines the following static property for the ConfigStore class:

- SchemaLocation

This section describes the behavior and requirements of this static method.

7.6.1 Schema Location

API Technology	Data Type	Access
.NET	string	R/O

Description

The full path of the Configuration Store's XML schema file. The schema is used when validating configuration store XML.

.NET Prototype

```
public static string SchemaLocation
```

7.7 IVI Configuration Store Static Methods (.NET Only)

The .NET IVI Configuration Store class defines the following static methods for the ConfigStore class:

- Load
- LoadWithoutValidation
- Validate

This section describes the behavior and requirements of each static method.

7.7.1 Load

Description

Create an instance of the ConfigStore class, deserialize the specified file, or the file at the master location if no file is specified, and return the instance. Validate the XML file against the Configuration Store's XML schema.

This method incurs the overhead of schema validation.

.NET Prototypes

```
public static ConfigStore Load(ConfigStoreLocation location =
                               ConfigStoreLocation.Master)


public static ConfigStore Load(string location)
```

7.7.2 Load Without Validation

Description

Create an instance of the ConfigStore class, deserialize the specified file, or the file at the master location if no file is specified, and return the instance. Do not validate the XML file against the Configuration Store's XML schema.

This method does not incur the overhead of schema validation.

.NET Prototype

```
public static ConfigStore LoadWithoutValidation (ConfigStoreLocation location =
                              ConfigStoreLocation.Master)


public static ConfigStore LoadWithoutValidation (string location)
```

7.7.3 Validate

Description

Validate the specified file against the Configuration Store's XML schema.

.NET Prototype

```
public static ConfigStore Validate (ConfigStoreLocation location =
                                    ConfigStoreLocation.Master)


public static ConfigStore Validate (string location)
```

8. IVI Hardware Asset Class

8.1 IVI Hardware Asset Overview

The IVI Hardware Asset class identifies the physical assets available to a system. Hardware assets are identified by their I/O Resource Descriptor, which generally speaking will be unique for a given instrument on a given PC.

The Data Components objects referenced by a hardware asset serve to document the hardware asset.

8.1.1 Documentation Data Components

Data components that document the hardware asset may be added to the hardware asset's data components collection at any time. These data components shall have Used In Session equal to "None" since they are not used to configure a software module.

8.2 IVI Hardware Asset Reference

The IVI Hardware Asset class inherits the following reference from IVI Configurable Component (Section5.3, *.NET API Special Features Overview*

This section defines special features of the .NET Configuration Server.

8.2.1 .NET Data Types

.NET uses the following basic data types in place of the corresponding COM types:

COM Data Type	.NET Data Type
BSTR	string
long	int
double	double
VARIANT_BOOL	bool

8.2.2 .NET Entity Class

Entity is an abstract base class that includes basic functionality used by other Config Store classes. Most of the class is not publically visible, but the class itself is visible and is used in several constructors including the main ConfigStore constructor.

The Entity class defines the following public property:

- Name

The following public Config Store classes are derived from Entity.

- HardwareAsset
- PublishedApi
- SoftwareModule
- PhysicalName
- PhysicalRange
- LogicalName
- Session
- DriverSession
- VirtualName

- VirtualRange
- DataComponent
- IviStructure
- IviInteger
- IviReal
- IviBoolean
- IviString
- IviAPIReference

For additional uses of the Entity class in .NET constructors, refer to 5.3.5.2, *Params Parameters in Constructors*.

8.2.2.1 The Entity.Name property.

The definition of Entity.Name matches the definition of the Name property in the classes that derive from entity.

In cases where a class places additional constriants on the Name property, the validation of Entity.Name is overridden by the class:

- Published Api
- IVI Physical Name
- IVI Virtual Name

8.2.3 .NET Enumerations

The .NET API defines the following enumerations:

- Config Store Location
- IVI Published API Name
- IVI Published API Type
- Session Usage

8.2.3.1 Config Store Location

The Config Store Location enumeration provides members for the two standard configuration store XML file locations.

Members do not have explicitly defined values.

Name	***Description***			
		Language	***Identifier***	***Value***
ConfigStoreLocation.Master	The master location.			
		.NET	`ConfigStoreLocation.Master`	
ConfigStoreLocation.ProcessDefault	The default location for the current process.			
		.NET	`ConfigStoreLocation.ProcessDefault`	

8.2.3.2 IVI Published API Name

The IVI Published API Name enumeration provides members for each of the standard APIs defined for IVI Instrument Drivers.

Members do not have explicitly defined values.

Name	***Description***			
		Language	***Identifier***	***Value***
IviPublishedApiName.IviDmm	The IviDmm instrument class.			
		.NET	`IviPublishedApiName.IviDmm`	
IviPublishedApiName.IviDriver	The IviDriver inherent capabilities class.			
		.NET	`IviPublishedApiName.IviDriver`	
IviPublishedApiName.IviScope	The IviScope instrument class.			
		.NET	`IviPublishedApiName.IviScope`	
IviPublishedApiName.IviFgen	The IviFgen instrument class.			
		.NET	`IviPublishedApiName.IviFgen`	
IviPublishedApiName.IviDCPwr	The IviDCPwr instrument class.			
		.NET	`IviPublishedApiName.IviDCPwr`	
IviPublishedApiName.IviACPwr	The IviACPwr instrument class.			
		.NET	`IviPublishedApiName.IviACPwr`	

IviPublishedApiName.IviSwtch	The IviSwtch instrument class.			
		.NET	`IviPublishedApiName.IviSwtch`	
IviPublishedApiName.IviPwrMeter	The IviPwrMeter instrument class.			
		.NET	`IviPublishedApiName.IviPwrMeter`	
IviPublishedApiName.IviSpecAn	The IviSpecAn instrument class.			
		.NET	`IviPublishedApiName.IviSpecAn`	
IviPublishedApiName.IviRFSigGen	The IviRFSigGen instrument class.			
		.NET	`IviPublishedApiName.IviRFSigGen`	
IviPublishedApiName.IviCounter	The IviCounter instrument class.			
		.NET	`IviPublishedApiName.IviCounter`	
IviPublishedApiName.IviDownConverter	The IviDownconverter instrument class.			
		.NET	`IviPublishedApiName.IviDownconverter`	
IviPublishedApiName.IviUpConverter	The IviUpconverter instrument class.			
		.NET	`IviPublishedApiName.IviUpconverter`	
IviPublishedApiName.IviDigitizer	The IviDigitizer instrument class.			
		.NET	`IviPublishedApiName.IviDigitizer`	
IviPublishedApiName.IviLxiSync	The IviLxiSync instrument class.			
		.NET	`IviPublishedApiName.IviLxiSync`	

8.2.3.3 IVI Published API Type

The IVI Published API Type enumeration provides members for the three standard API types defined for IVI Instrument Drivers.

Members do not have explicitly defined values.

Name	***Description***			
		Language	***Identifier***	***Value***
IviPublishedApiType.IviCom	The IVI-COM API type.			
		.NET	`IviPublishedApiType.IviCom`	
IviPublishedApiType.IviNet	The IVI.NET API type.			
		.NET	`IviPublishedApiType.IviNet`	
IviPublishedApiType.IviC	The IVI-C API type.			
		.NET	`IviPublishedApiType.IviC`	

8.2.3.4 Session Usage

The Config Store Location enumeration provides members for the three standard values for data component Used In Session properties.

Name	*Description*			
		Language	*Identifier*	*Value*
SessionUsage.None	A session does not need to define a value for this data component.			
		.NET	`SessionUsage.None`	`0`
SessionUsage. Required	A session must define a value for this data component.			
		.NET	`SessionUsage.Required`	`1`
SessionUsage. Optional	A session may define a value for this data component.			
		.NET	`SessionUsage.Optional`	`2`

8.2.4 Collections in .NET

Refer to section 4.5, *Collections in .NET* for a description of collections in .NET.

8.2.5 .NET Constructors

In contrast to the COM API which does not allow parameterized constructors, the .NET API uses parameterized constructors, so that new objects have legal and consistent data from the start.

8.2.5.1 Constructors and Property Access

Each constructor includes parameters for

- Properties that serve as keys in the corresponding collection. For example, Name is the key value for nearly all classes, and so name is a common constructor parameter.
- Properties that should not be changed after they are set initially. For example, the Prefix property for SoftwareModules only needs to be set once, and does not need to be changed after that.

In general, if a constructor includes a parameter, the corresponding property is read only.

8.2.5.2 Params Parameters in Constructors

Some constructors include a params parameter. In the .NET API, these allow a calling program to specify members of the object's collections. For example, the constructor for the Virtual Name class includes the `virtualRanges` parameter, which allows the calling program to specify multiple Virtual Range objects to be added to the Virtual Name's Virtual Range collection.

In most cases the type of the params parameter is an array of a base type, either Entity or DataComponent. This allows the calling program to specify any object whose class derives from the base class. The constructor is then responsible for ensuring that the specified objects are valid for that constructor. For example, the Config Store constructor has a params parameter of type Entity[]. This allows the calling program to specify any class that derives from Entity as an argument. However, the Config Store constructor checks each argument to make sure that they are valid for the constructor, and only six types are valid: Driver Sessions, Hardware Assets, Logical Names, Published APIs, Sessions, and Software Modules. If the calling program specifies an object of an invalid derived type, the constructor will throw an exception.

Constructors may enforce additional order constraints on the specified objects. The most common constraint is for items that must be added to one of the six global collections before they are referenced elsewhere. If an order constraint is violated, the constructor will throw an exception.

8.2.5.3 Collection Constructors

Collection constructors are not public. Instead, the Configuration Server creates collections when needed, and allows calling programs to populate the collections using the collection Add() methods or constructor params arguments.

8.2.6 .NET Static Methods and Properties

The .NET API includes a few static methods and properties. These are documented for the classes where they are defined, in separate sections to emphasize the fact that they are static. For the most part these methods and properties do not have an exact match in the C or COM APIs.

The most important of the static methods is the factory method Load(…). Load combines instantiating a ConfigStore object and deserializing a configuration store file, returning the instantiated and loaded object. In contract, the ConfigStore constructor should only be used to create a new configuration store from scratch.

8.2.7 .NET Schema Validation

The .NET API provides more control over validation than the C or COM APIs. The default for all of the APIs is to validate an XML file against the Configuration Store XML schema when a file is deserialized. In addition, the .NET API allows a calling program to deserialize a file without validation (for performance) and to validate a file without deserializing it.

8.2.8 .NET Exceptions

Refer to section 5.12.2, *IVI.NET Error Handling*, of *IVI-3.1, Driver Architecture Specification*, for a general overview of exceptions in IVI.NET components. Calling programs should be able to accommodate arbitrary exceptions, including those defined by the .NET Framework.

The .NET Configuration Server does not define any new exceptions. When it explicitly throws an exception, it throws one of the standard .NET exceptions (usually ArgumentException or InvalidOperationException) with a custom message that indicates the problem.

The Configuration Server does not generate warnings.

IVI Configurable Components Class (Virtual)):
- Data Components

8.3 IVI Hardware Asset Properties

The IVI Hardware Asset class defines the following property:
- I/O Resource Descriptor

The IVI Hardware Asset class inherits the following properties from IVI Configurable Component (Section5.3, *.NET API Special Features Overview*

This section defines special features of the .NET Configuration Server.

8.3.1 .NET Data Types

.NET uses the following basic data types in place of the corresponding COM types:

COM Data Type	.NET Data Type
BSTR	string
long	int

double	double
VARIANT_BOOL	bool

8.3.2 .NET Entity Class

Entity is an abstract base class that includes basic functionality used by other Config Store classes. Most of the class is not publically visible, but the class itself is visible and is used in several constructors including the main ConfigStore constructor.

The Entity class defines the following public property:
- Name

The following public Config Store classes are derived from Entity.
- HardwareAsset
- PublishedApi
- SoftwareModule
- PhysicalName
- PhysicalRange
- LogicalName
- Session
- DriverSession
- VirtualName
- VirtualRange
- DataComponent
- IviStructure
- IviInteger
- IviReal
- IviBoolean
- IviString
- IviAPIReference

For additional uses of the Entity class in .NET constructors, refer to 5.3.5.2, *Params Parameters in Constructors*.

8.3.2.1 The Entity.Name property.

The definition of Entity.Name matches the definition of the Name property in the classes that derive from entity.

In cases where a class places additional constriants on the Name property, the validation of Entity.Name is overridden by the class:
- Published Api
- IVI Physical Name
- IVI Virtual Name

8.3.3 .NET Enumerations

The .NET API defines the following enumerations:

- Config Store Location
- IVI Published API Name
- IVI Published API Type
- Session Usage

8.3.3.1 Config Store Location

The Config Store Location enumeration provides members for the two standard configuration store XML file locations.

Members do not have explicitly defined values.

Name	***Description***			
		Language	***Identifier***	***Value***
ConfigStoreLocation.Master	The master location.			
		.NET	`ConfigStoreLocation.Master`	
ConfigStoreLocation.ProcessDefault	The default location for the current process.			
		.NET	`ConfigStoreLocation.ProcessDefault`	

8.3.3.2 IVI Published API Name

The IVI Published API Name enumeration provides members for each of the standard APIs defined for IVI Instrument Drivers.

Members do not have explicitly defined values.

Name	***Description***			
		Language	***Identifier***	***Value***
IviPublishedApiName.IviDmm	The IviDmm instrument class.			
		.NET	`IviPublishedApiName.IviDmm`	
IviPublishedApiName.IviDriver	The IviDriver inherent capabilities class.			
		.NET	`IviPublishedApiName.IviDriver`	
IviPublishedApiName.IviScope	The IviScope instrument class.			
		.NET	`IviPublishedApiName.IviScope`	
IviPublishedApiName.IviFgen	The IviFgen instrument class.			
		.NET	`IviPublishedApiName.IviFgen`	
IviPublishedApiName.IviDCPwr	The IviDCPwr instrument class.			
		.NET	`IviPublishedApiName.IviDCPwr`	
IviPublishedApiName.IviACPwr	The IviACPwr instrument class.			
		.NET	`IviPublishedApiName.IviACPwr`	

IviPublishedApiName.IviSwtch	The IviSwtch instrument class.			
	.NET	`IviPublishedApiName.IviSwtch`	
IviPublishedApiName.IviPwrMeter	The IviPwrMeter instrument class.			
	.NET	`IviPublishedApiName.IviPwrMeter`	
IviPublishedApiName.IviSpecAn	The IviSpecAn instrument class.			
	.NET	`IviPublishedApiName.IviSpecAn`	
IviPublishedApiName.IviRFSigGen	The IviRFSigGen instrument class.			
	.NET	`IviPublishedApiName.IviRFSigGen`	
IviPublishedApiName.IviCounter	The IviCounter instrument class.			
	.NET	`IviPublishedApiName.IviCounter`	
IviPublishedApiName.IviDownConverter	The IviDownconverter instrument class.			
	.NET	`IviPublishedApiName.IviDownconverter`	
IviPublishedApiName.IviUpConverter	The IviUpconverter instrument class.			
	.NET	`IviPublishedApiName.IviUpconverter`	
IviPublishedApiName.IviDigitizer	The IviDigitizer instrument class.			
	.NET	`IviPublishedApiName.IviDigitizer`	
IviPublishedApiName.IviLxiSync	The IviLxiSync instrument class.			
	.NET	`IviPublishedApiName.IviLxiSync`	

8.3.3.3 IVI Published API Type

The IVI Published API Type enumeration provides members for the three standard API types defined for IVI Instrument Drivers.

Members do not have explicitly defined values.

Name	***Description***			
		Language	***Identifier***	***Value***
IviPublishedApiType.IviCom	The IVI-COM API type.			
	.NET	`IviPublishedApiType.IviCom`	
IviPublishedApiType.IviNet	The IVI.NET API type.			
	.NET	`IviPublishedApiType.IviNet`	
IviPublishedApiType.IviC	The IVI-C API type.			
	.NET	`IviPublishedApiType.IviC`	

8.3.3.4 Session Usage

The Config Store Location enumeration provides members for the three standard values for data component Used In Session properties.

Name	*Description*			
		Language	*Identifier*	*Value*
SessionUsage.None	A session does not need to define a value for this data component.			
		.NET	`SessionUsage.None`	`0`
SessionUsage. Required	A session must define a value for this data component.			
		.NET	`SessionUsage.Required`	`1`
SessionUsage. Optional	A session may define a value for this data component.			
		.NET	`SessionUsage.Optional`	`2`

8.3.4 Collections in .NET

Refer to section 4.5, *Collections in .NET* for a description of collections in .NET.

8.3.5 .NET Constructors

In contrast to the COM API which does not allow parameterized constructors, the .NET API uses parameterized constructors, so that new objects have legal and consistent data from the start.

8.3.5.1 Constructors and Property Access

Each constructor includes parameters for

- Properties that serve as keys in the corresponding collection. For example, Name is the key value for nearly all classes, and so name is a common constructor parameter.
- Properties that should not be changed after they are set initially. For example, the Prefix property for SoftwareModules only needs to be set once, and does not need to be changed after that.

In general, if a constructor includes a parameter, the corresponding property is read only.

8.3.5.2 Params Parameters in Constructors

Some constructors include a params parameter. In the .NET API, these allow a calling program to specify members of the object's collections. For example, the constructor for the Virtual Name class includes the `virtualRanges` parameter, which allows the calling program to specify multiple Virtual Range objects to be added to the Virtual Name's Virtual Range collection.

In most cases the type of the params parameter is an array of a base type, either Entity or DataComponent. This allows the calling program to specify any object whose class derives from the base class. The constructor is then responsible for ensuring that the specified objects are valid for that constructor. For example, the Config Store constructor has a params parameter of type Entity[]. This allows the calling program to specify any class that derives from Entity as an argument. However, the Config Store constructor checks each argument to make sure that they are valid for the constructor, and only six types are valid: Driver Sessions, Hardware Assets, Logical Names, Published APIs, Sessions, and Software Modules. If the calling program specifies an object of an invalid derived type, the constructor will throw an exception.

Constructors may enforce additional order constraints on the specified objects. The most common constraint is for items that must be added to one of the six global collections before they are referenced elsewhere. If an order constraint is violated, the constructor will throw an exception.

8.3.5.3 Collection Constructors

Collection constructors are not public. Instead, the Configuration Server creates collections when needed, and allows calling programs to populate the collections using the collection Add() methods or constructor params arguments.

8.3.6 .NET Static Methods and Properties

The .NET API includes a few static methods and properties. These are documented for the classes where they are defined, in separate sections to emphasize the fact that they are static. For the most part these methods and properties do not have an exact match in the C or COM APIs.

The most important of the static methods is the factory method Load(…). Load combines instantiating a ConfigStore object and deserializing a configuration store file, returning the instantiated and loaded object. In contract, the ConfigStore constructor should only be used to create a new configuration store from scratch.

8.3.7 .NET Schema Validation

The .NET API provides more control over validation than the C or COM APIs. The default for all of the APIs is to validate an XML file against the Configuration Store XML schema when a file is deserialized. In addition, the .NET API allows a calling program to deserialize a file without validation (for performance) and to validate a file without deserializing it.

8.3.8 .NET Exceptions

Refer to section 5.12.2, *IVI.NET Error Handling*, of *IVI-3.1, Driver Architecture Specification*, for a general overview of exceptions in IVI.NET components. Calling programs should be able to accommodate arbitrary exceptions, including those defined by the .NET Framework.

The .NET Configuration Server does not define any new exceptions. When it explicitly throws an exception, it throws one of the standard .NET exceptions (usually ArgumentException or InvalidOperationException) with a custom message that indicates the problem.

The Configuration Server does not generate warnings.

IVI Configurable Components Class (Virtual)):

- Description
- Name

This section describes the behavior and requirements of the property defined in the IVI Hardware class.

8.3.9 I/O Resource Descriptor

API Technology	Data Type	Access
.NET	string	R/W
C	ViString	R/W
COM	BSTR	R/W

COM/.NET Property Name

IOResourceDescriptor

C Constant Name

IVICONFIG_VAL_HARDWARE_ASSET_IO_DESCRIPTOR

Description

I/O Resource Descriptor stores a string that specifies the address of the hardware asset that can be recognized by I/O used by a software module that will access the hardware. For a complete description of the I/O Resource Descriptor property, refer to Section 5.17, *Function Compliance Rules* and Section 6.14, *Initialize*, of the *IVI-3.2: Inherent Capabilities Specification*.

The empty string is a legal value for this property.

8.4 IVI Hardware Asset Constructor (.NET Only)

The .NET IVI Hardware Asset class defines one public constructor.

This section describes the behavior and requirements of the constructor.

8.4.1 HardwareAsset Constructor

Description

Creates an instance of a hardware asset.

DataComponents parameters may be any one of the following classes deriving from DataComponent:

- IviStructure,
- IviInteger
- IviReal
- IviBoolean
- IviString
- IviAPIReference

DataComponents parameters are added to the Harware Asset's Data Components Collection in the order in which they are specified.

The IVI Published Api referenced by a DataComponents IviAPIReference parameter must have been added to the global Published API Collection before the Hardware Asset is added to the global Hardware Asset Collection.

.NET Prototype

```
public HardwareAsset(string name, string ioResourceDescriptor,
                     params DataComponent[] dataComponents)
```

.NET Parameters

Inputs	Description	.NET Type
`name`	The name of the Hardware Asset. This name must be unique in any collection of Hardware Assets which includes this one.	`string`
`ioResourceDescriptor`	The I/O Resource Descriptor of the asset.	`string`
`dataComponents`	Zero to n Data Components to be referenced by this Hardware Asset.	`DataComponent[]`

9. IVI Published API Class

9.1 IVI Published API Overview

Published APIs are APIs that are supported across several products, and as such are published independently of any one component that implements them. Published APIs may be standard APIs published by the IVI Foundation, such as the IviDriver API for IVI driver inherent capabilities (refer to *IVI-3.2: Inherent Capabilities Specification*) or the instrument class specification APIs. Published APIs may also be IVI-MSS roles or IVI Signal Interface APIs. Published APIs may also be vendor-defined APIs that span multiple components.

Published APIs are identified by name. The name is a logical description of the API. For instance, the IVI-C IviDriver API consists of function prototypes, attribute ID constants, and so on, while the IVI-COM IviDriver API consists of IDL enumerations and interface definitions. In both cases, the name of the published API is "IviDriver". Published APIs are not necessarily tied to any particular revision of an API. Suppose that the IviDriver API is revised, so that both version 1 and version 2 are implemented in different drivers. In both cases, the name of the published API is "IviDriver". Published API names are specified by the document that specifies the API.

For IVI Instrument Classes, the published API name shall be the instrument class name.

The Type field identifies the syntax of the API. "IVI.NET", "IVI-C", and "IVI-COM" are predefined values that shall be used for IVI defined APIs. There will be separate Published API entries in the configuration server for the IVI.NET IviDriver API, the IVI-C IviDriver API, and the IVI-COM IviDriver API.

The Version fields identify revisions of the API specification which are reflected in the API. Multiple revisions of a specific API (identified by Name and Type) may be stored in the configuration store. For IviDriver and IVI class specifications, the major and minor versions shall be the major and minor versions of the specification supported by the driver.

Together, the Name, Type, and Version properties identify a unique entry in the global Published API collection. This uniqueness is enforced by the configuration server. However, the properties appear individually in the API.

9.2 IVI Published API Properties

The IVI Published API class defines the following properties:

- Major Version
- Minor Version
- Name
- Type

This section describes the behavior and requirements of each property.

9.2.1 Major Version

API Technology	Data Type	Access
.NET	int	R/O
C	ViInt32	R/W
COM	long	R/W

COM/.NET Property Name

MajorVersion

C Constant Name

IVICONFIG_VAL_PUBLISHED_API_MAJOR_VERSION

Description

The major version of this revision of the published API. This is determined by the person or group who publishes the API.

9.2.2 Minor Version

API Technology	Data Type	Access
.NET	int	R/O
C	ViInt32	R/W
COM	long	R/W

COM/.NET Property Name

MinorVersion

C Constant Name

IVICONFIG_VAL_PUBLISHED_API_MINOR_VERSION

Description

The minor version of this revision of the published API. This is determined by the person or group who publishes the API.

9.2.3 Name

API Technology	Data Type	Access
.NET	string	R/O
C	ViString	R/W
COM	BSTR	R/W

COM/.NET Property Name

Name

C Constant Name

IVICONFIG_VAL_PUBLISHED_API_NAME

Description

The name of a Published API. Name may refer to either:

- An IVI-defined API. In this case, the first 3 characters of the string shall be “Ivi”. These API names are defined in the various IVI specifications.
- An API defined outside of the IVI Foundation. In this case, the first three characters of the string shall not be “Ivi”, where “Ivi” is case insensitive. The definition of these names is specific to the person or group that defines the API.

The empty string is not a legal value for this property.

9.2.4 Type

API Technology	Data Type	Access
.NET	string	R/O
C	ViString	R/W
COM	BSTR	R/W

COM/.NET Property Name

Type

C Constant Name

IVICONFIG_VAL_PUBLISHED_API_TYPE

Description

The type of a Published API.

Three predefined values are supported for the IVI defined interfaces, “IVI.NET”, “IVI-C”, and “IVI-COM”. These values shall be used for all IVI defined inherent and class APIs.

Other user-defined values may be used for other types of APIs. However, if the APIs follow the style for IVI.NET, IVI-C, or IVI-COM interfaces as described in *IVI-3.4: API Style Guide*, it is recommended that “IVI.NET”, “IVI-C”, or “IVI-COM” be used as the type, respectively.

The empty string is not a legal value for this property.

9.3 IVI Published API Static Methods (.NET Only)

The IVI Configuration Store class defines the following methods to assist the calling program in translating between `IviPublishedApiName` and string, and `IviPublishedApiType` and string:

- ParseName
- ParseType
- TryParseName
- TryParseType
- TypeToString

This section describes the behavior and requirements of each method.

9.3.1 Parse Name

Description

Converts the specified Published API name class name to the corresponding enum value for an IVI defined Published API. If there is not a corresponding IVI defined name for the specified Published API name, the method throws an ArgumentException with an explanatory message.

To avoid handling an exception, use this method only if you are sure that the specified Published API name is an IVI defined name. If you are not sure, use the Try Parse Name method instead. Refer to section 9.3.3, *Try Parse Name* for details.

.NET Prototype

```
public static IviPublishedApiName ParseName(string name)
```

.NET Parameters

Inputs	Description	.NET Type
`name`	The Published API name.	`string`

Return Value	Description	.NET Type
	The IVI defined Published API name.	`IviPublishedApiName`

9.3.2 Parse Type

Description

Converts the specified Published API type name to the corresponding enum value for an IVI defined Published API type. If there is not a corresponding IVI defined type for the specified Published API type name, the method throws an ArgumentException with an explanatory message

To avoid handling an exception, use this method only if you are sure that the specified Published API type name is an IVI defined Published API type. If you are not sure, use the Try Parse Type method instead. Refer to section 0, *.NET Parameters*

Inputs	Description	.NET Type
`name`	The name of the Published API.	`string`

Outputs	Description	.NET Type
`iviName`	The IVI defined Published API name that corresponds to the specified `name`, if one exists. If one does not exist, this value is undefined.	`IviPublishedApiName`

Return Value	Description	.NET Type
	True if an IVI defined name was found for the specified name, otherwise false.	`bool`

Try Parse Type for details.

.NET Prototype

```
public static IviPublishedApiType ParseType(string typeName)
```

.NET Parameters

Inputs	Description	.NET Type
`typeName`	The name of the Published API Type.	`string`

Return Value	Description	.NET Type
	The IVI defined Published API name.	`IviPublishedApiType`

9.3.3 Try Parse Name

Description

Converts the specified Published API name to the corresponding enum value for an IVI defined Published API type. If there is not a corresponding IVI defined name for the specified name, this function returns False and the iviName output parameter is undefined.

.NET Method Prototype

```
public static bool TryParseName(string name, out IviPublishedApiName iviName)
```

.NET Parameters

Inputs	Description	.NET Type
`name`	The name of the Published API.	`string`

Outputs	Description	.NET Type
`iviName`	The IVI defined Published API name that corresponds to the specified `name`, if one exists. If one does not exist, this value is undefined.	`IviPublishedApiName`

Return Value	Description	.NET Type
	True if an IVI defined name was found for the specified name, otherwise false.	`bool`

9.3.4 Try Parse Type

Description

Converts the specified Published API type name to the corresponding enum value for an IVI defined Published API type. If there is not a corresponding IVI defined type for the specified Published API type name, this function returns False and the type output parameter is undefined.

.NET Prototype

```
public static bool TryParseType(string typeName, out IviPublishedApiType type)
```

.NET Parameters

Inputs	Description	.NET Type
`typeName`	The name of the Published API Type..	`string`

Outputs	Description	.NET Type
`type`	The IVI defined type that corresponds to the specified typeName, if one exists. If one does not exist, this value is undefined.	`IviPublishedApiType`

Return Value	Description	.NET Type
	True if an IVI defined type was found for the Published API Type, otherwise false.	`bool`

9.3.5 Type To String

Description

Converts the specified IVI defined Published API type to a string.

.NET Method Prototype

```
public static string TypeToString(IviPublishedApiType type)
```

.NET Parameters

Inputs	Description	.NET Type
type	The name of an IVI defined Published API.	string

Return Value	Description	.NET Type
	The string form of the name.	bool

9.4 IVI Published API Constructors (.NET Only)

The .NET IVI Published API class defines two public constructors.

The first constructor takes arbitrary values for name and type, and has the flexibility to describe published APIs that are defined by software module vendors for modules that are not defined by the IVI Foundation. This constructors paramaters more closely match the data types of the corresponding .NET properties.

The second constructor takes an enumerated value for name and type, and has the advantage of limiting the calling program to a list of valid values for IVI instrument drivers when that is what is intended.

This section describes the behavior and requirements of the constructors.

9.4.1 PublishedAPI Constructors

Description

Creates an instance of an Ivi Published Api.

.NET Prototype

```
public PublishedApi(string name,
                    string type,
                    int majorVersion,
                    int minorVersion)

public PublishedApi(IviPublishedApiName name,
                    IviPublishedApiType type,
                    int majorVersion,
                    int minorVersion)
```

.NET Parameters

Inputs	Description	.NET Type
`name`	The name of any Ivi Published Api.	`string`
	The name of an Ivi Published Api that is supported by the IVI Foundation for IVI specific drivers,	`IviPublishedApiName`
`type`	The type of any Ivi Published Api.	`string`
	The type of an Ivi Published Api that is supported by the IVI Foundation for IVI specific drivers,	`IviPublishedApiType`
`majorVersion`	The major version of the revision of the published API.	`int`
`minorVersion`	The minor version of the revision of the published API.	`int`

9.5 IVI Published API Collection Methods and Operators (.NET Only)

All of the IVI configuration server collections have Name as the unique key value, with the exception of the IVI Published API Collection. Each item in a IVI Published API Collection is uniquely identified by a combination of four values: Name, Type, Major Version, and Minor Version. As a result, operations on the IVI Published API collection take four parameters to represent the key, rather than the one parameter used in corresponding operations for other collections.

The public IVI Published API Collection methods and operators affected by the key requirements are:

- [] – The indexer for the collection.
- Contains Key
- Remove
- TryGetValue

There are two forms of each method/operator. The first form allows for general definition of Published APIs by taking string values for name and type parameters. The second form allows for validated definition of IVI defined APIs by taking IviPublishedAPIName values for the name parameter and IviPublishedAPIType values for the type parameter.

10. IVI Software Module Class

10.1 IVI Software Module Overview

The Software Module class identifies IVI software components. IVI software components include, but are not limited to, various types of IVI drivers and IVI-MSS components.

A Software Module is a software component that exposes its functionality via an ANSI C, COM, or .NET API, or both. For ANSI C modules, the value of the Module Path property is the full pathname or simple filename of the DLL containing the ANSI C entry points. For COM modules, the value of the ProgID property is the version independent ProgID of the software module's COM coclass. If a single vendor supports both ANSI C and COM APIs for the Software Module object, both Module Path and ProgID have valid values. The Software Module installer is responsible for making sure one or both of these entries are not empty. If both entries are empty, the client program cannot find the Software Module's executable code.

.NET software modules are always in separate SoftwareModules. For .NET software modules, the AssemblyQualifiedClassName is used to instrantiate the driver. ModulePath and ProgID are not used. The Software Module installer is responsible for making sure this entry is not empty. If it is empty, the client program cannot find the Software Module's executable code. The name of a .NET software module for IVI.NET specific instrument drivers shall be *<Namespace>.<FwkVerShortName>*.

Software Modules may expose one or more APIs that are published and implemented by one or more modules. IVI class-compliant interfaces are examples of such APIs. Refer to Section 9.1, *IVI Published API Overview* for a description of how Published APIs are defined and treated in the Configuration Server. If a software module implements both ANSI C and COM APIs for the same module, the associated Published API collection shall include references to both types of the APIs.

Software Modules may implement repeated capabilities. Refer to Section 2.8.7. *IVI API Reference* for a description of how repeated capabilities are defined in the Configuration Server. If a Software Module implements repeated capabilities, if must reference a collection of IVI Physical Name objects. The IVI Physical Name objects may be structured hierarchically to reflect a hierarchy of repeated capabilities in the Software Module. Refer to Section 11.1, *IVI Physical Name Overview* for an overview of how IVI Physical Names may be structured hierarchically.

The Data Components objects referenced by a software module are either configurable initial settings or serve to document the software module.

10.1.1 Configurable Initial Settings

A configurable initial setting in an IVI Software Module is an IVI Data Component object with particular characteristics. An IVI Software Module object's IVI Data Components collection may contain an IVI Structure whose name is "Configurable Initial Settings". An IVI Data Component in this IVI Structure with Used In Session equal to "Required" or "Optional", Read Only equal to "True", and Type equal to "Boolean", "String", "Integer", "Real", or "APIReference" is a configurable initial setting. An IVI Structure is never a configurable initial setting. The Value property of the data component is a suitable default value for the setting when the software module is initialized, if possible. The Description property should contain a description of the configurable initial setting and its possible values. If this exceeds the practical size limit for a string, then the data component's help properties should refer the user to a suitable form of help that fully describes the configurable initial setting and its possible values. Refer to Section 5.5.3, *Defining Configurable Initial Settings in the IVI Configuration Store*, of *IVI-3.17: Installation Requirements Specification*, for more information on the Configurable Initial Settings structure.

Because the configurable initial settings are selected by the software module developer, and are recognized by the software module during initialization, configurable initial settings are only added by the software module when it is installed.

Configurable initial settings from the software module are copied to sessions that reference the software module. Refer to Section 14.1.1, *Configurable Initial Settings*, for more details.

10.1.2 Documentation Data Components

Data components that document the software module in some way may be added to the software module's data components collection at any time. These data components shall have Used In Session equal to "None" since they are not used by the session to configure the software module.

10.1.3 API Type-Specific Software Module Classes (.NET Only)

The .NET Configration Server includes three additional classes that are specific to each of the three API types supported by IVI.

- IviComSoftwareModule
- IviNetSoftwareModule
- IviCSoftwareModule

These classes derive from SoftwareModule. They are designed to make it clear what properties are used to create new instances of each type of module.

The SoftwareModule class exposes all properties used to find and load modules: ProgID. Module Path, Module Path 32, Module Path 64, and Assembly Qualified Class Name. The derived classes only expose the properties that are used to find and load modules of that API type.

10.1.3.1 IviComSoftwareModule

COM software modules are located using the ProgID property.

If a COM software module includes a C wrapper, the module is also located using Module Path, Module Path 32, and Module Path 64 properties.

The IviComSoftwareModule class constructor includes ProgID. Module Path 32, and Module Path 64, but not Assembly Qualified Class Name.

The IviComSoftwareModule class exposes the ProgID. Module Path, Module Path 32, and Module Path 64 properties, but not the Assembly Qualified Class Name property.

10.1.3.2 IviCSoftwareModule

C software modules are located using Module Path, Module Path 32, and Module Path 64 properties.

The IviCSoftwareModule class constructor includes Module Path 32, and Module Path 64, but not ProgID or Assembly Qualified Class Name.

The IviCSoftwareModule class exposes Module Path, Module Path 32, and Module Path 64 properties, but not the ProgID or Assembly Qualified Class Name properties.

10.1.3.3 IviNetSoftwareModule

.NET software modules are located using the Assembly Qualified Class Name property.

The IviNetSoftwareModule class constructor includes Assembly Qualified Class Name, but not ProgID. Module Path, Module Path 32, or Module Path 64.

The IviNetSoftwareModule class exposes the Assembly Qualified Class Name property, but not the ProgID. Module Path, Module Path 32, or Module Path 64 properties.

10.2 IVI Software Module References

The IVI Software Module class defines the following references:

- Physical Names
- Published APIs

The IVI Software Module class inherits the following references from IVI Configurable Component - Section 5.3, *.NET API Special Features Overview*

This section defines special features of the .NET Configuration Server.

10.2.1 .NET Data Types

.NET uses the following basic data types in place of the corresponding COM types:

COM Data Type	.NET Data Type
BSTR	string
long	int
double	double
VARIANT_BOOL	bool

10.2.2 .NET Entity Class

Entity is an abstract base class that includes basic functionality used by other Config Store classes. Most of the class is not publically visible, but the class itself is visible and is used in several constructors including the main ConfigStore constructor.

The Entity class defines the following public property:

- Name

The following public Config Store classes are derived from Entity.

- HardwareAsset
- PublishedApi
- SoftwareModule
- PhysicalName
- PhysicalRange
- LogicalName
- Session
- DriverSession
- VirtualName
- VirtualRange
- DataComponent
- IviStructure
- IviInteger
- IviReal
- IviBoolean
- IviString
- IviAPIReference

For additional uses of the Entity class in .NET constructors, refer to 5.3.5.2, *Params Parameters in Constructors*.

10.2.2.1The Entity.Name property.

The definition of Entity.Name matches the definition of the Name property in the classes that derive from entity.

In cases where a class places additional constriants on the Name property, the validation of Entity.Name is overridden by the class:

- Published Api
- IVI Physical Name
- IVI Virtual Name

10.2.3 .NET Enumerations

The .NET API defines the following enumerations:

- Config Store Location
- IVI Published API Name
- IVI Published API Type
- Session Usage

10.2.3.1Config Store Location

The Config Store Location enumeration provides members for the two standard configuration store XML file locations.

Members do not have explicitly defined values.

Name	***Description***			
		Language	***Identifier***	***Value***
ConfigStoreLocation.Master	The master location.			
		.NET	`ConfigStoreLocation.Master`	
ConfigStoreLocation.ProcessDefault	The default location for the current process.			
		.NET	`ConfigStoreLocation.ProcessDefault`	

10.2.3.2IVI Published API Name

The IVI Published API Name enumeration provides members for each of the standard APIs defined for IVI Instrument Drivers.

Members do not have explicitly defined values.

Name	***Description***			
		Language	***Identifier***	***Value***
IviPublishedApiName.IviDmm	The IviDmm instrument class.			
		.NET	`IviPublishedApiName.IviDmm`	
IviPublishedApiName.IviDriver	The IviDriver inherent capabilities class.			
		.NET	`IviPublishedApiName.IviDriver`	
IviPublishedApiName.IviScope	The IviScope instrument class.			
		.NET	`IviPublishedApiName.IviScope`	
IviPublishedApiName.IviFgen	The IviFgen instrument class.			
		.NET	`IviPublishedApiName.IviFgen`	
IviPublishedApiName.IviDCPwr	The IviDCPwr instrument class.			
		.NET	`IviPublishedApiName.IviDCPwr`	
IviPublishedApiName.IviACPwr	The IviACPwr instrument class.			
		.NET	`IviPublishedApiName.IviACPwr`	

IviPublishedApiName.IviSwtch	The IviSwtch instrument class.			
		.NET	`IviPublishedApiName.IviSwtch`	
IviPublishedApiName.IviPwrMeter	The IviPwrMeter instrument class.			
		.NET	`IviPublishedApiName.IviPwrMeter`	
IviPublishedApiName.IviSpecAn	The IviSpecAn instrument class.			
		.NET	`IviPublishedApiName.IviSpecAn`	
IviPublishedApiName.IviRFSigGen	The IviRFSigGen instrument class.			
		.NET	`IviPublishedApiName.IviRFSigGen`	
IviPublishedApiName.IviCounter	The IviCounter instrument class.			
		.NET	`IviPublishedApiName.IviCounter`	
IviPublishedApiName.IviDownConverter	The IviDownconverter instrument class.			
		.NET	`IviPublishedApiName.IviDownconverter`	
IviPublishedApiName.IviUpConverter	The IviUpconverter instrument class.			
		.NET	`IviPublishedApiName.IviUpconverter`	
IviPublishedApiName.IviDigitizer	The IviDigitizer instrument class.			
		.NET	`IviPublishedApiName.IviDigitizer`	
IviPublishedApiName.IviLxiSync	The IviLxiSync instrument class.			
		.NET	`IviPublishedApiName.IviLxiSync`	

10.2.3.3IVI Published API Type

The IVI Published API Type enumeration provides members for the three standard API types defined for IVI Instrument Drivers.

Members do not have explicitly defined values.

Name	***Description***			
		Language	***Identifier***	***Value***
IviPublishedApiType.IviCom	The IVI-COM API type.			
		.NET	`IviPublishedApiType.IviCom`	
IviPublishedApiType.IviNet	The IVI.NET API type.			
		.NET	`IviPublishedApiType.IviNet`	
IviPublishedApiType.IviC	The IVI-C API type.			
		.NET	`IviPublishedApiType.IviC`	

10.2.3.4Session Usage

The Config Store Location enumeration provides members for the three standard values for data component Used In Session properties.

Name	*Description*			
		Language	*Identifier*	*Value*
SessionUsage.None	A session does not need to define a value for this data component.			
		.NET	`SessionUsage.None`	`0`
SessionUsage. Required	A session must define a value for this data component.			
		.NET	`SessionUsage.Required`	`1`
SessionUsage. Optional	A session may define a value for this data component.			
		.NET	`SessionUsage.Optional`	`2`

10.2.4 Collections in .NET

Refer to section 4.5, *Collections in .NET* for a description of collections in .NET.

10.2.5 .NET Constructors

In contrast to the COM API which does not allow parameterized constructors, the .NET API uses parameterized constructors, so that new objects have legal and consistent data from the start.

10.2.5.1Constructors and Property Access

Each constructor includes parameters for

- Properties that serve as keys in the corresponding collection. For example, Name is the key value for nearly all classes, and so name is a common constructor parameter.
- Properties that should not be changed after they are set initially. For example, the Prefix property for SoftwareModules only needs to be set once, and does not need to be changed after that.

In general, if a constructor includes a parameter, the corresponding property is read only.

10.2.5.2Params Parameters in Constructors

Some constructors include a params parameter. In the .NET API, these allow a calling program to specify members of the object's collections. For example, the constructor for the Virtual Name class includes the `virtualRanges` parameter, which allows the calling program to specify multiple Virtual Range objects to be added to the Virtual Name's Virtual Range collection.

In most cases the type of the params parameter is an array of a base type, either Entity or DataComponent. This allows the calling program to specify any object whose class derives from the base class. The constructor is then responsible for ensuring that the specified objects are valid for that constructor. For example, the Config Store constructor has a params parameter of type Entity[]. This allows the calling program to specify any class that derives from Entity as an argument. However, the Config Store constructor checks each argument to make sure that they are valid for the constructor, and only six types are valid: Driver Sessions, Hardware Assets, Logical Names, Published APIs, Sessions, and Software Modules. If the calling program specifies an object of an invalid derived type, the constructor will throw an exception.

Constructors may enforce additional order constraints on the specified objects. The most common constraint is for items that must be added to one of the six global collections before they are referenced elsewhere. If an order constraint is violated, the constructor will throw an exception.

10.2.5.3Collection Constructors

Collection constructors are not public. Instead, the Configuration Server creates collections when needed, and allows calling programs to populate the collections using the collection Add() methods or constructor params arguments.

10.2.6 .NET Static Methods and Properties

The .NET API includes a few static methods and properties. These are documented for the classes where they are defined, in separate sections to emphasize the fact that they are static. For the most part these methods and properties do not have an exact match in the C or COM APIs.

The most important of the static methods is the factory method Load(…). Load combines instantiating a ConfigStore object and deserializing a configuration store file, returning the instantiated and loaded object. In contract, the ConfigStore constructor should only be used to create a new configuration store from scratch.

10.2.7 .NET Schema Validation

The .NET API provides more control over validation than the C or COM APIs. The default for all of the APIs is to validate an XML file against the Configuration Store XML schema when a file is deserialized. In addition, the .NET API allows a calling program to deserialize a file without validation (for performance) and to validate a file without deserializing it.

10.2.8 .NET Exceptions

Refer to section 5.12.2, *IVI.NET Error Handling*, of *IVI-3.1, Driver Architecture Specification*, for a general overview of exceptions in IVI.NET components. Calling programs should be able to accommodate arbitrary exceptions, including those defined by the .NET Framework.

The .NET Configuration Server does not define any new exceptions. When it explicitly throws an exception, it throws one of the standard .NET exceptions (usually ArgumentException or InvalidOperationException) with a custom message that indicates the problem.

The Configuration Server does not generate warnings.

IVI Configurable Components Class (Virtual)

- Data Components

This section describes each reference.

10.2.9 Physical Names

API Technology	Data Type	Access
COM	IIviPhysicalNameCollection**	R/O
C	IviPhysicalNameCollectionHandle	R/O
.NET	PhysicalNameCollection	R/O

COM/.NET Property Name

```
PhysicalNames
```

C Function Prototype

```
ViStatus _VI_FUNC IviConfig_GetSoftwareModulePhysicalNameCollection
                                                (IviSoftwareModuleHandle
                                                SoftwareModuleHandle,
                                                IviPhysicalNameCollectionHandle*
                                                PhysicalNameCollectionHandle);
```

C Parameters

Inputs	Description	Datatype
SoftwareModuleHandle	Handle to an IviSoftwareModule object.	IviSoftwareModuleHandle

Outputs	Description	Datatype
PhysicalNameCollectionHandle	Handle to an IviPhysicalNameCollection object.	IviPhysicalNameCollectionHandle

Description

References a collection of all the PhysicalName objects defined by the Software Module.

If a Software Module implements IVI instrument class-compliant APIs that have repeated capabilities, it shall reference a collection of IVI Physical Name objects for the class-defined repeated capabilities. This information is required in order to specify virtual identifiers for IVI Sessions and Driver Sessions, and virtual identifiers are required to use repeated capabilities interchangeably.

For other repeated capabilities in a Software Module, physical identifiers are optional but recommended, particularly if needed to support interchangeability.

The IVI Physical Name collections may be structured hierarchically to reflect a hierarchy of repeated capabilities in the Software Module. Refer to Section 11.1, *IVI Physical Name Overview* for an description of how physical identifiers may be structured hierarchically.

10.2.10 Published APIs

API Technology	Data Type	Access
COM	IIviPublishedAPICollection**	R/O
C	IviPublishedAPICollectionHandle	R/O
.NET	PublishedAPICollection	R/O

COM/.NET Property Name

PublishedAPIs

C Function Prototype

```
ViStatus _VI_FUNC IviConfig_GetSoftwareModulePublishedAPIsCollection
                                                    (IviSoftwareModuleHandle
                                                    SoftwareModuleHandle,
                                                    IviPublishedAPIsCollectionHandle*
                                                    PublishedAPIsCollectionHandle);
```

C Parameters

Inputs	Description	Datatype
SoftwareModuleHandle	Handle to an IviSoftwareModule object.	IviSoftwareModuleHandle

Outputs	Description	Datatype
PublishedAPIsCollectionHandle	Handle to an IviPublishedAPIsCollection object.	IviPublishedAPIsCollectionHandle

Description

References a collection of all the Published APIs that are implemented by the Software Module. Refer to Section 9.1, *IVI Published API Overview* for an overview of Published APIs.

Before a Published API can be added to a Software Module's Published API collection, it must already exist in the global Published API collection. To add a Published API to a Software Module's Published API collection first get the Published API from the global collection and the add the returned reference to the Software Module's collection.

10.3 IVI Software Module Properties

The IVI Software Module class defines the following properties:

- Assembly Qualified Class Name
- Module Path
- Module Path 32
- Module Path 64
- Prefix
- ProgID
- Supported Instrument Models\

The IVI Software Module class inherits the following properties from IVI Configurable Component - Section 5.3, *.NET API Special Features Overview*

This section defines special features of the .NET Configuration Server.

10.3.1 .NET Data Types

.NET uses the following basic data types in place of the corresponding COM types:

COM Data Type	.NET Data Type
BSTR	string
long	int
double	double
VARIANT_BOOL	bool

10.3.2 .NET Entity Class

Entity is an abstract base class that includes basic functionality used by other Config Store classes. Most of the class is not publically visible, but the class itself is visible and is used in several constructors including the main ConfigStore constructor.

The Entity class defines the following public property:

- Name

The following public Config Store classes are derived from Entity.

- HardwareAsset
- PublishedApi
- SoftwareModule
- PhysicalName
- PhysicalRange
- LogicalName
- Session
- DriverSession
- VirtualName
- VirtualRange
- DataComponent
- IviStructure
- IviInteger
- IviReal
- IviBoolean
- IviString
- IviAPIReference

For additional uses of the Entity class in .NET constructors, refer to 5.3.5.2, *Params Parameters in Constructors*.

10.3.2.1 The Entity.Name property.

The definition of Entity.Name matches the definition of the Name property in the classes that derive from entity.

In cases where a class places additional constriants on the Name property, the validation of Entity.Name is overridden by the class:

- Published Api
- IVI Physical Name
- IVI Virtual Name

10.3.3 .NET Enumerations

The .NET API defines the following enumerations:

- Config Store Location
- IVI Published API Name
- IVI Published API Type
- Session Usage

10.3.3.1Config Store Location

The Config Store Location enumeration provides members for the two standard configuration store XML file locations.

Members do not have explicitly defined values.

Name	***Description***			
		Language	***Identifier***	***Value***
ConfigStoreLocation.Master	The master location.			
		.NET	`ConfigStoreLocation.Master`	
ConfigStoreLocation.ProcessDefault	The default location for the current process.			
		.NET	`ConfigStoreLocation.ProcessDefault`	

10.3.3.2IVI Published API Name

The IVI Published API Name enumeration provides members for each of the standard APIs defined for IVI Instrument Drivers.

Members do not have explicitly defined values.

Name	***Description***			
		Language	***Identifier***	***Value***
IviPublishedApiName.IviDmm	The IviDmm instrument class.			
		.NET	`IviPublishedApiName.IviDmm`	
IviPublishedApiName.IviDriver	The IviDriver inherent capabilities class.			
		.NET	`IviPublishedApiName.IviDriver`	
IviPublishedApiName.IviScope	The IviScope instrument class.			
		.NET	`IviPublishedApiName.IviScope`	
IviPublishedApiName.IviFgen	The IviFgen instrument class.			
		.NET	`IviPublishedApiName.IviFgen`	
IviPublishedApiName.IviDCPwr	The IviDCPwr instrument class.			
		.NET	`IviPublishedApiName.IviDCPwr`	
IviPublishedApiName.IviACPwr	The IviACPwr instrument class.			
		.NET	`IviPublishedApiName.IviACPwr`	

IviPublishedApiName.IviSwtch	The IviSwtch instrument class.			
		.NET	`IviPublishedApiName.IviSwtch`	
IviPublishedApiName.IviPwrMeter	The IviPwrMeter instrument class.			
		.NET	`IviPublishedApiName.IviPwrMeter`	
IviPublishedApiName.IviSpecAn	The IviSpecAn instrument class.			
		.NET	`IviPublishedApiName.IviSpecAn`	
IviPublishedApiName.IviRFSigGen	The IviRFSigGen instrument class.			
		.NET	`IviPublishedApiName.IviRFSigGen`	
IviPublishedApiName.IviCounter	The IviCounter instrument class.			
		.NET	`IviPublishedApiName.IviCounter`	
IviPublishedApiName.IviDownConverter	The IviDownconverter instrument class.			
		.NET	`IviPublishedApiName.IviDownconverter`	
IviPublishedApiName.IviUpConverter	The IviUpconverter instrument class.			
		.NET	`IviPublishedApiName.IviUpconverter`	
IviPublishedApiName.IviDigitizer	The IviDigitizer instrument class.			
		.NET	`IviPublishedApiName.IviDigitizer`	
IviPublishedApiName.IviLxiSync	The IviLxiSync instrument class.			
		.NET	`IviPublishedApiName.IviLxiSync`	

10.3.3.3IVI Published API Type

The IVI Published API Type enumeration provides members for the three standard API types defined for IVI Instrument Drivers.

Members do not have explicitly defined values.

Name	***Description***			
		Language	***Identifier***	***Value***
IviPublishedApiType.IviCom	The IVI-COM API type.			
		.NET	`IviPublishedApiType.IviCom`	
IviPublishedApiType.IviNet	The IVI.NET API type.			
		.NET	`IviPublishedApiType.IviNet`	
IviPublishedApiType.IviC	The IVI-C API type.			
		.NET	`IviPublishedApiType.IviC`	

10.3.3.4Session Usage

The Config Store Location enumeration provides members for the three standard values for data component Used In Session properties.

Name	*Description*			
		Language	*Identifier*	*Value*
SessionUsage.None	A session does not need to define a value for this data component.			
		.NET	`SessionUsage.None`	`0`
SessionUsage. Required	A session must define a value for this data component.			
		.NET	`SessionUsage.Required`	`1`
SessionUsage. Optional	A session may define a value for this data component.			
		.NET	`SessionUsage.Optional`	`2`

10.3.4 Collections in .NET

Refer to section 4.5, *Collections in .NET* for a description of collections in .NET.

10.3.5 .NET Constructors

In contrast to the COM API which does not allow parameterized constructors, the .NET API uses parameterized constructors, so that new objects have legal and consistent data from the start.

10.3.5.1Constructors and Property Access

Each constructor includes parameters for

- Properties that serve as keys in the corresponding collection. For example, Name is the key value for nearly all classes, and so name is a common constructor parameter.
- Properties that should not be changed after they are set initially. For example, the Prefix property for SoftwareModules only needs to be set once, and does not need to be changed after that.

In general, if a constructor includes a parameter, the corresponding property is read only.

10.3.5.2Params Parameters in Constructors

Some constructors include a params parameter. In the .NET API, these allow a calling program to specify members of the object's collections. For example, the constructor for the Virtual Name class includes the `virtualRanges` parameter, which allows the calling program to specify multiple Virtual Range objects to be added to the Virtual Name's Virtual Range collection.

In most cases the type of the params parameter is an array of a base type, either Entity or DataComponent. This allows the calling program to specify any object whose class derives from the base class. The constructor is then responsible for ensuring that the specified objects are valid for that constructor. For example, the Config Store constructor has a params parameter of type Entity[]. This allows the calling program to specify any class that derives from Entity as an argument. However, the Config Store constructor checks each argument to make sure that they are valid for the constructor, and only six types are valid: Driver Sessions, Hardware Assets, Logical Names, Published APIs, Sessions, and Software Modules. If the calling program specifies an object of an invalid derived type, the constructor will throw an exception.

Constructors may enforce additional order constraints on the specified objects. The most common constraint is for items that must be added to one of the six global collections before they are referenced elsewhere. If an order constraint is violated, the constructor will throw an exception.

10.3.5.3Collection Constructors

Collection constructors are not public. Instead, the Configuration Server creates collections when needed, and allows calling programs to populate the collections using the collection Add() methods or constructor params arguments.

10.3.6 .NET Static Methods and Properties

The .NET API includes a few static methods and properties. These are documented for the classes where they are defined, in separate sections to emphasize the fact that they are static. For the most part these methods and properties do not have an exact match in the C or COM APIs.

The most important of the static methods is the factory method Load(…). Load combines instantiating a ConfigStore object and deserializing a configuration store file, returning the instantiated and loaded object. In contract, the ConfigStore constructor should only be used to create a new configuration store from scratch.

10.3.7 .NET Schema Validation

The .NET API provides more control over validation than the C or COM APIs. The default for all of the APIs is to validate an XML file against the Configuration Store XML schema when a file is deserialized. In addition, the .NET API allows a calling program to deserialize a file without validation (for performance) and to validate a file without deserializing it.

10.3.8 .NET Exceptions

Refer to section 5.12.2, *IVI.NET Error Handling*, of *IVI-3.1, Driver Architecture Specification*, for a general overview of exceptions in IVI.NET components. Calling programs should be able to accommodate arbitrary exceptions, including those defined by the .NET Framework.

The .NET Configuration Server does not define any new exceptions. When it explicitly throws an exception, it throws one of the standard .NET exceptions (usually ArgumentException or InvalidOperationException) with a custom message that indicates the problem.

The Configuration Server does not generate warnings.

IVI Configurable Components Class (Virtual)

- Description
- Name

This section describes the behavior and requirements of each property defined in the IVI Software Module class.

10.3.9 Assembly Qualified Class Name

API Technology	Data Type	Access
.NET	string	R/O
C	ViString	R/W
COM	BSTR	R/W

COM/.NET Property Name

AssemblyQualifiedClassName

C Constant Name

IVICONFIG_VAL_SW_MODULE_ASMBLY_QLFD_CLASS_NAME

Description

Returns a string that is the assembly qualified class name of the default .NET class of the software module. The assembly-qualified name of a type consists of the type name, including its namespace, followed by a comma, followed by the four-part display name of the assembly. The display name includes the simple name of the assembly, a version number, a cryptographic key pair, and a supported culture[1].

For .NET software modules the following line of code returns the assembly qualified class name (where `module` is a reference to the software module's main, instantiable class):

```
module.GetType().AssemblyQualifiedName;
```

This property shall be filled in for .NET software modules. It shall be an empty string for C or COM software modules.

[1] Note that processor architecture is included in assembly identity in .NET 2.0, but is not included in the assembly qualified name.

10.3.10 Module Path

API Technology	Data Type	Access
.NET	string	None
C	ViString	R/W
COM	BSTR	R/W

COM/.NET Property Name

ModulePath

C Constant Name

IVICONFIG_VAL_SW_MODULE_PATH

Description

Returns a string that is either the simple file name or the full pathname of the software module DLL. When running in a native 32-bit context, ModulePath returns ModulePath32. When running in a native 64-bit context, ModulePath returns ModulePath64. This property may be an empty string.

For backwards compatibility with earlier versions of the Configuration Server, ModulePath sets ModulePath32 when running in a native 32-bit context. When running in a native 64-bit context, attempts to set ModulePath return a Not Supported.

Note that ModulePath as defined here is not represented in the version 1.6 schema at all. ModulePath in the version 1.6 schema actually stores the value of the Configuration Server's ModulePath32.

10.3.11 Module Path 32

API Technology	Data Type	Access
.NET	string	R/O
C	ViString	R/W
COM	BSTR	R/W

COM/.NET Property Name

ModulePath32

C Constant Name

IVICONFIG_VAL_SW_MODULE_PATH_32

Description

Returns a string that is either the simple filename or the full pathname of the native 32-bit software module DLL. This property may be an empty string.

Note that ModulePath32 is represented in the schema as ModulePath, rather than ModulePath32, to preserve the backwards compatibility with data files created by versions of the Configuration Server prior to version 1.6.

10.3.12 Module Path 64

API Technology	Data Type	Access
.NET	string	R/O
C	ViString	R/W
COM	BSTR	R/W

COM/.NET Property Name

ModulePath64

C Constant Name

IVICONFIG_VAL_SW_MODULE_PATH_64

Description

Returns a string that is either the simple filename or the full pathname of the native 64-bit software module DLL. This property may be an empty string.

10.3.13 Prefix

API Technology	Data Type	Access
.NET	string	R/O
C	ViString	R/W
COM	BSTR	R/W

COM/.NET Property Name

Prefix

C Constant Name

IVICONFIG_VAL_SW_MODULE_PREFIX

Description

Prefix is a string that specifies the prefix (for IVI-C components) or the component identifier (for IVI-COM and IVI.NET components) of the software module. This shall exactly match the value returned by the Prefix attribute or the Component Identifier attributes. Refer to Section 5.6, *Class Driver Prefix*, Section 5.13, *Component Identifier*, and Section 5.32, *Specific Driver Prefix*, of the *IVI-3.2: Inherent Capabilities Specification*, for complete details.

The empty string is a legal value for this property.

10.3.14 ProgID

API Technology	Data Type	Access
.NET	string	R/O
C	ViString	R/W
COM	BSTR	R/W

COM/.NET Property Name

ProgID

C Constant Name

IVICONFIG_VAL_SW_MODULE_PROGID

Description

ProgID returns a string that specifies the version-independent COM ProgID of the software module. This property may be an empty string.

10.3.15 Supported Instrument Models

API Technology	Data Type	Access
.NET	string	R/W
C	ViString	R/W
COM	BSTR	R/W

COM/.NET Property Name

SupportedInstrumentModels

C Constant Name

IVICONFIG_VAL_SW_MODULE_SUPPORTED_INSTR_MODELS

Description

A comma-separated string that specifies the instrument models supported by the software module. This shall exactly match the value returned by the Supported Instrument Models attribute as defined in Section 5.35, *Supported Instrument Models*, of the *IVI-3.2: Inherent Capabilities Specification*.

10.4 IVI Software Module Constructors (.NET Only)

The .NET IVI Software Module class defines one public constructor.

The .NET IVI COM Software Module class defines two public constructors.

The .NET IVI C Software Module class defines one public constructor.

The .NET IVI .NET Software Module class defines one public constructor.

This section describes the behavior and requirements of each constructor.

10.4.1 SoftwareModule Constructor

Description

Creates an instance of a Software Module. ProgID, Module Path 32, Module Path64, and Assembly Qualified Path Name are not specified.

Children parameters may be any one of the following classes deriving from Entity:

- PublishedAPI
- PhysicalName
- DataComponents

If there are no children parameters, the SoftwareModule is created without any Physical Name or Published API references.

Children parameters are added to the corresponding collections in the order in which they are specified.

Before a Published API object may be added to the Software Module's Published APIs collection, that Published API object must have already been added to the global Published APIs Collection.

.NET Prototype

```
public SoftwareModule(string name, string prefix, params Entity[] children)
```

.NET Parameters

Inputs	Description	.NET Type
`name`	The name of the Software Module.	`string`
`prefix`	The Prefix or Component Indentifier of the Software Module.	`string`
`children`	Zero to n Published APIs and zero to n Physical Names to be referenced by the Software Module.	`params Entity[]`

10.4.2 IviComSoftwareModule Constructor

Description

Creates an instance of an IVI-COM Software Module.

There are two overloads of the constructor:
- The first specifies ProgID, and describes a Software Module with a COM instrument driver with no C wrapper included in the same DLL.
- The second specifies ProgID, Module Path 32, and Module Path64, and describes a Software Module with a COM instrument driver with a C wrapper included in the same DLL.

Children parameters may be any one of the following classes deriving from Entity:
- PublishedAPI
- PhysicalName

If there are no children parameters, the IVI-COM Software Module is created without any Physical Name or Published API references.

Children parameters are added to the corresponding collections in the order in which they are specified.

If a Published API object is included in the Software Module's Published APIs collection, that Published API object must have already been added to the global Published APIs Collection.

.NET Prototype

```
public IviComSoftwareModule(string name,
                            string prefix,
                            string progId,
                            params Entity[] children)

public IviComSoftwareModule(string name,
                            string prefix,
                            string progId,
                            string modulePath32,
                            string modulePath64,
                            params Entity[] children)
```

.NET Parameters

Inputs	Description	.NET Type
`name`	The name of the Software Module.	`string`
`prefix`	The Prefix or Component Indentifier of the Software Module.	`string`
`progId`	A string that specifies the version-independent COM ProgID of the COM software module.	`string`
`modulePath32`	For COM modules that include C wrappers, either the simple filename or the full pathname of the native 32-bit software module DLL. For software modules that do not include a C wrapper or modules that do not include a native 32-bit wrapper, this is the empty string.	`string`

modulePath64	For COM modules that include C wrappers, either the simple filename or the full pathname of the native 64-bit software module DLL. For software modules that do not include a C wrapper or modules that do not include a native 64-bit wrapper, this is the empty string.	string
children	Zero to n Published APIs and zero to n Physical Names to be referenced by the Software Module.	params Entity[]

10.4.3 IviCSoftwareModule Constructor

Description

Creates an instance of an IVI-C Software Module and specifies Module Path 32 and Module Path64.

Children parameters may be any one of the following classes deriving from Entity:

- PublishedAPI
- PhysicalName,

If there are no children parameters, the IVI-C Software Module is created without any Physical Name or Published API references.

Children parameters are added to the corresponding collections in the order in which they are specified.

If a Published API object is included in the Software Module's Published APIs collection, that Published API object must have already been added to the global Published APIs Collection.

.NET Prototype

```
public IviCSoftwareModule(string name,
                          string prefix,
                          string modulePath32,
                          string modulePath64,
                          params Entity[] children)
```

.NET Parameters

Inputs	Description	.NET Type
`name`	The name of the Software Module.	`string`
`prefix`	The Prefix or Component Indentifier of the Software Module.	`string`
`modulePath32`	The full pathname of the native 32-bit software module DLL. For software modules that do not include a native 32-bit executable, this is the empty string.	`string`
`modulePath64`	The full pathname of the native 64-bit software module DLL. For software modules that do not include a native 64-bit executable, this is the empty string.	`string`
`children`	Zero to n Published APIs and zero to n Physical Names to be referenced by the Software Module.	`params Entity[]`

10.4.4 IviNetSoftwareModule Constructor

Description

Creates an instance of an IVI.NET Software Module and specifies the Assembly Qualified Class Name.

Children parameters may be any one of the following classes deriving from Entity:

- PublishedAPI
- PhysicalName,

If there are no children parameters, the IVI.NET Software Module is created without any Physical Name or Published API references.

Children parameters are added to the corresponding collections in the order in which they are specified.

If a Published API object is included in the Software Module's Published APIs collection, that Published API object must have already been added to the global Published APIs Collection.

.NET Prototype

```
public IviNetSoftwareModule(string name,
                            string assemblyQualifiedPathName,
                            string prefix,
                            params Entity[] children)
```

.NET Parameters

Inputs	Description	.NET Type
`name`	The name of the Software Module.	`string`
`prefix`	The Prefix or Component Indentifier of the Software Module.	`string`
`assemblyQualifiedPathName`	The assembly qualified class name of the default .NET class of the software module.	`string`
`children`	Zero to n Published APIs and zero to n Physica Names to be referenced by the Software Module.	`params Entity[]`

11. IVI Physical Name Class

11.1 IVI Physical Name Overview

IVI Physical Name objects describe the physical identifiers supported by a software module. Physical identifiers are the names that an IVI specific instrument driver, IVI-MSS role control module, or other IVI software module gives to instances of the repeated capabilities they implement. For example, one IviScope specific instrument driver might name channels “A”, “B”, and “C” while another might name them “1”, “2”, “3’, and “4”.

The RC Name property describes the type of repeated capability. In the above example, RC Name might be “Channel” – the name of the repeated capability in the IviScope specification.

An IVI Physical Name object can reference a collection of IVI Physical Name objects, thereby creating a hierarchy of nested repeated capabilities.

An IVI Physical Name object can reference a collection of IVI Physical Range objects, which allows an efficient way of creating a large number of physical identifiers by appending integers from the range to the Name property.

For a comprehensive overview of the treatment of repeated capabilities in the Configuration Server, including the role that physical identifiers play, refer to Section 2.9, *Repeated Capabilities*.

11.2 IVI Physical Name References

The IVI Physical Name class defines the following references:

- Physical Names
- Physical Ranges

This section describes the behavior and requirements of each property.

11.2.1 Physical Names

API Technology	Data Type	Access
COM	IIviPhysicalNameCollection**	R/O
C	IviPhysicalNameCollectionHandle	R/O
.NET	PhysicalNameCollection	R/O

COM/.NET Property Name

PhysicalNames

C Function Prototype

```
ViStatus _VI_FUNC IviConfig_GetPhysicalNameChildPhysicalNameCollection
                                                (IviPhysicalNameHandle
                                                PhysicalNameHandle,
                                                IviPhysicalNameCollectionHandle*
                                                ChildPhysicalNameCollectionHandle);
```

C Parameters

Inputs	Description	Datatype
PhysicalNameHandle	Handle to an IviPhysicalName object.	IviPhysicalNameHandle

Outputs	Description	Datatype
ChildPhysicalNameCollectionHandle	Handle to an IviPhysicalNameCollection object.	IviPhysicalNameCollectionHandle

Description

References a collection of the IVI Physical Name objects that are hierarchically structured under this IVI Physical Name object. This collection will commonly be empty.

11.2.2 Physical Ranges

API Technology	Data Type	Access
COM	IIviPhysicalRangeCollection**	R/O
C	IviPhysicalRangeCollectionHandle	R/O
.NET	PhysicalRangeCollection	R/O

COM/.NET Property Name

```
PhysicalRanges
```

C Function Prototype

```
ViStatus _VI_FUNC IviConfig_GetPhysicalNamePhysicalRangeCollection
                                                (IviPhysicalNameHandle
                                                PhysicalNameHandle,
                                                IviPhysicalRangeCollectionHandle*
                                                PhysicalRangeCollectionHandle);
```

C Parameters

Inputs	Description	Datatype
PhysicalNameHandle	Handle to an IviPhysicalName object.	IviPhysicalNameHandle

Outputs	Description	Datatype
PhysicalRangeCollectionHandle	Handle to an IviPhysicalRangeCollection object.	IviPhysicalRangeCollectionHandle

Description

References a collection of IVI Physical Range objects used to qualify the referencing IVI Physical Name object.

11.3 IVI Physical Name Properties

The IVI Physical Name class defines the following properties:

- Name
- RC Name

This section describes the behavior and requirements of each property.

11.3.1 Name

API Technology	Data Type	Access
.NET	string	R/O
C	ViString	R/W
COM	BSTR	R/W

COM/.NET Property Name

Name

C Constant Name

IVICONFIG_VAL_PHYSICAL_NAME_NAME

Description

The name of a specific instance of a repeated capability as defined in the software module. This is also known as the physical identifier.

Name shall consist of one or more of the following characters: ‘a’-‘z’, ‘A’-‘Z’, ‘0’-‘9’, ‘!’, and ‘_’. In cases where driver writers need to qualify the physical name with the repeated capability name or qualified repeated capability name, “!!” is used to separate the repeated capability name from the portion of the name that designates the repeated capability instance. For example if a driver implements both an IviScope Channel and an IviDigitizer Channel called “C1”, then the physical name for the scope channel would be “IviScopeChannel!!C1” where the instance of the repeated capability name, “C1”, is disambiguated from the Digitizer channel “C1” by prefixing “IviScopeChannel” followed by “!!”. Beginning January 1, 2009, “!!” shall be reserved for this purpose in the Name property.

The empty string is valid for this property only if the IVI Physical Name object references a non-empty collection of Physical Range objects. Note that since Name is a key for the IVI Physical Name collection, only one Name per collection may be empty.

11.3.2 RC Name

API Technology	Data Type	Access
.NET	string	R/W
C	ViString	R/W
COM	BSTR	R/W

COM/.NET Property Name

RCName

C Constant Name

IVICONFIG_VAL_PHYSICAL_NAME_RCNAME

Description

The repeated capability name as defined in the software module. Each IVI Physical Name object represents an instance of the repeated capability of type RC Name. For software modules that reference Published APIs that define repeated capabilities, the RC Name shall be the repeated capability name defined by the Published API specification.

A single driver repeated capability may be used to implement two or more Published API repeated capabilities. (For example, a driver that implements both the IviScope and IviDigitizer instrument classes may combine the IviScope "Channel" and IviDigitizer "Channel" repeated capabilities into a single repeated capability.) In such cases, it is possible that the names of the repeated capabilities defined by the Published APIs will be different, and if they are, RC Name shall be one of the names defined by the Published APIs.

The empty string is not a legal value for this property.

11.4 IVI Physical Name Constructor (.NET Only)

The .NET IVI Physical Name class defines one public constructor.

This section describes the behavior and requirements of the constructor.

11.4.1 PhysicalName Constructors

Description

Creates an instance of a Physical Name.

Children parameters may be any one of the following classes deriving from Entity:
- PhysicalName,
- PhysicalRange

If there are no children parameters, the Physical Name is created without any Physical Name or Physical Range references.

Children parameters are added to the corresponding collections in the order in which they are specified.

.NET Prototype

```
public PhysicalName(string name, params Entity[] children)
```

.NET Parameters

Inputs	Description	.NET Type
`name`	The name of the PhysicalName.	`string`
`children`	Zero to n Physical Names and zero to n Physical Ranges to be referenced by the Physical Name	`params Entity[]`

12. IVI Physical Range Class

12.1 IVI Physical Range Overview

The IVI Physical Range class allows multiple instances of a repeated capability to be defined with a minimum of effort. An IVI Physical Range object shall be referenced by exactly one IVI Physical Name object.

For a comprehensive overview of the treatment of repeated capabilities in the Configuration Server, including the role that physical names play, refer to Section 2.9, *Repeated Capabilities*.

12.2 IVI Physical Range Properties

The IVI Physical Range class defines the following properties:

- Max
- Min
- Name

This section describes the behavior and requirements of each property.

12.2.1 Max

API Technology	Data Type	Access
.NET	int	R/W
C	ViInt32	R/W
COM	long	R/W

COM/.NET Property Name

Max

C Constant Name

IVICONFIG_VAL_PHYSICAL_RANGE_MAX

Description

The upper end of a range of integers to be appended to the Name property of the referencing IVI Physical Name object to create a set of physical repeated capability identifiers.

12.2.2 Min

API Technology	Data Type	Access
.NET	int	R/W
C	ViInt32	R/W
COM	long	R/W

COM/.NET Property Name

Min

C Constant Name

IVICONFIG_VAL_PHYSICAL_RANGE_MIN

Description

The lower end of a range of integers to be appended to the Name property of the referencing IVI Physical Name object to create a set of physical repeated capability identifiers.

12.2.3 Name

API Technology	Data Type	Access
.NET	string	R/O
C	ViString	R/W
COM	BSTR	R/W

COM/.NET Property Name

Name

C Constant Name

IVICONFIG_VAL_PHYSICAL_RANGE_NAME

Description

The name of the physical range. This name is used to uniquely identify the range in the collection, and is not used in creating the set of physical identifiers.

The empty string is not a legal value for this property.

12.3 IVI Physical Range Constructors (.NET Only)

The .NET IVI Physical Range class defines one public constructor.

This section describes the behavior and requirements of the constructor.

12.3.1 PhysicalRange Constructors

Description

Creates an instance of a Physical Range.

.NET Prototype

```
public PhysicalRange(string name, int min, int max)
```

.NET Parameters

Inputs	Description	.NET Type
`name`	The name of the Physical Range.	`string`
`min`	The lower bound of a range of integers to be appended to the Name property of the referencing IVI PhysicalName object to create a set of physical repeated capability identifiers.	`int`
`max`	The upper bound of a range of integers to be appended to the Name property of the referencing IVI PhysicalName object to create a set of physical repeated capability identifiers.	`int`

13. IVI Logical Name Class

13.1 IVI Logical Name Overview

Logical Names introduce an additional level of indirection when referencing IVI Sessions and IVI Driver Sessions. Logical Names allow users to define and name multiple Sessions and switch between them by referencing a Logical Name in a client program. The user changes the Logical Name reference to point to any one of the Sessions depending on the situation, without changing source code.

It is impossible to tell by looking at a Logical Name object whether the reference is to a Session or a Driver Session. The Session object must be examined by the client code to determine whether or not it is a Driver Session.

13.2 IVI Logical Name Reference

The IVI Logical Name class defines the following reference:

- Session

This section describesthe reference.

13.2.1 Session

API Technology	Data Type	Access
COM	IIviSession**	R/O
C	IviSessionHandle	R/O
.NET	Session	R/O

COM/.NET Property Name

```
Session
```

C Function Prototype

```
ViStatus _VI_FUNC IviConfig_GetLogicalNameSessionReference
                                                (IviLogicalNameHandle
                                                LogicalNameHandle,
                                                IviSessionHandle*
                                                SessionHandle);

ViStatus _VI_FUNC IviConfig_SetLogicalNameSessionReference
                                               (IviLogicalNameHandle
                                               LogicalNameHandle,
                                               IviSessionHandle SessionHandle);
```

C Parameters

Inputs	Description	Datatype
LogicalNameHandle	Handle to an IviLogicalNameHandle object.	IviLogicalNameHandle
SessionHandle	Handle to an IviSession object.	IviSessionHandle

Outputs	Description	Datatype
SessionHandle	Handle to an IviSession object.	IviSessionHandle

Description

The IVI Session to which the logical name refers. The IVI Session may be an IVI Driver Session.

Return Values

The *IVI-3.2: Inherent Capabilities Specification* defines general status codes that this function can return.

The table below specifies additional IVI configuration server status codes for this function.

Completion Codes	Description
Not In Global Collection	The item does not exist in the global collection.

13.3 IVI Logical Name Properties

The IVI Logical Name class defines the following properties:

- Name

This section describes the behavior and requirements of each property.

13.3.1 Name

API Technology	Data Type	Access
.NET	string	R/O
C	ViString	R/W
COM	BSTR	R/W

COM/.NET Property Name

Name

C Constant Name

IVICONFIG_VAL_LOGICAL_NAME_NAME

Description

The logical name.

The empty string is not a legal value for this property.

13.3.2 Description

API Technology	Data Type	Access
.NET	string	R/W
C	ViString	R/W
COM	BSTR	R/W

COM/.NET Property Name

Description

C Constant Name

IVICONFIG_VAL_CONFIG_COMPONENT_DESCRIPTION

Description

The description of the logical name. The empty string is a legal value for this property.

13.4 IVI Logical Name Constructors (.NET Only)

The .NET IVI Logical Name class defines one public constructor.

This section describes the behavior and requirements of the constructor.

13.4.1 LogicalName Constructors

Description

Creates an instance of a Logical Name.

If a Session object is referenced by the Logical Name, that Session object must have already been added to the global Session Collection.

.NET Prototype

```
public LogicalName(string name, Session session = null)
```

.NET Parameters

Inputs	Description	.NET Type
`name`	The name of the Logical Name.	`string`
`session`	A reference to a Session object that is already a member of the parent ConfigStore's Sessions collection, or null.	`Session`

14. IVI Session Class

14.1 IVI Session Overview

The IVI Session class provides the information needed to configure a software module. A software module may be referenced by several sessions. In other words, the configuration store may contain several configurations for one software module. The client specifies at run time which configuration will be used.

The Software Module property refers to the specific module configured by an IVI Session object.

The Hardware Asset property refers to the hardware asset that identifies the specific instrument or other physical asset that will be used by the software module.

Virtual identifiers are used to assign client specific names to the physical identifiers used by the software module for repeated capabilities. Refer to Section 16.1, *IVI Virtual Name Overview*, for more information.

The Data Components objects referenced by a session are either configurable initial settings or serve to document the session.

14.1.1 Configurable Initial Settings

Certain IVI Data Components, including configurable initial settings, in a software module are copied to the sessions that reference the software module.

A transferable data component is an IVI Data Component in an IVI Software Module whose type is not Structure with Used In Session equal to "Required" or "Optional". It may be directly in the software module's IVI Data Components collection. It may also be in an IVI Structure whose name is "Configurable Initial Settings" which is in the software module's IVI Data Components collection. Refer to Section 5.5.3, *Defining Configurable Initial Settings in the IVI Configuration Store*, of *IVI-3.17: Installation Requirements Specification*, for more information on this structure.

A transferred data component in an IVI Session is a transferable data component which has been copied. It has the same characteristics as a transferable data component, though its Read Only property is set to "False". Its Value property is copied from the matching transferable data component, but it may be changed by the user. This Value is used by the software module when it is initialized. All of the other properties are copied over unchanged from the software module referenced by the session.

A session shall include copies of all of the transferable data components with Used In Session equal to "Required" from the referenced software module. Clients may optionally copy transferable data components with Used In Session equal to "Optional" from the referenced software module. Before any transferable data component in the "Configurable Initial Settings" structure is copied, an equivalent IVI Structure is first added to the session and the transferred data component appears in that IVI Structure.

In general, the configuration server tries to preserve a session and all of its transferred data components, even if the referenced software module is deleted. This preserves configuration information when, for instance, a driver is reinstalled or upgraded to a new version. The configuration server employs relatively complex logic to meet this objective.

- When the session's Software Module reference property is set for the first time, the configuration server copies all of the transferable data components with Used In Session equal to "Required" from the referenced software module. Users may subsequently copy transferable data components with Used In Session equal to "Optional" over to the session using a configuration utility.
- When the session's Software Module reference property is set to a null reference, the configuration server deletes all of the transferred data components from the session. The session does not "remember" the previously referenced software module by its Name.

- When the session's Software Module reference is changed explicitly, the configuration server deletes all of the transferred data components from the session then copies transferable data components as if the reference had been set for the first time.
- If the software module referenced by the session's Software Module reference property is deleted, the session's Software Module reference property is set to a null reference implicitly. However, the session will "remember" the previously referenced software module by its Name. This remembered name is returned by the Software Module Name property. The configuration server does not delete the session's transferred data components in this case.
- When the configuration server adds a new software module, it finds all sessions with Software Module Name identical to that of the newly added software module. For each session found, the configuration server checks to see if the session's transferred data components match the newly added software module's transferable data components.
  - A software module's transferable data component matches a session's transferred data component if the two have identical values for the Name, Type, and, where applicable, Units properties.
  - If a match is found, then the values for Description, Help Context ID, Help File Path, Software Module Key, and Used In Session are copied from the software module's transferable data component to the matching session's transferred data component. Read Only and Value are not changed in the session's transferred data component.
  - Software module transferable data components that don't match any session transferred data components are copied to the session if Used In Session is equal to "Required".
  - Session transferred data components that don't match any software module transferable data components are deleted from the session if Used In Session is equal to "Required".

14.1.2 Documentation Data Components

Data components that document the session in some way may be added to the session's data components collection at any time. These data components shall have "Used In Session" equal to "None" since they are not used by the session to configure the software module.

14.2 IVI Session References

The IVI Session class defines the following references:
- Hardware Asset
- Software Module
- Virtual Names

The IVI Software Module class inherits the following reference from IVI Configurable Component - Section 5.3, *.NET API Special Features Overview*

This section defines special features of the .NET Configuration Server.

14.2.1 .NET Data Types

.NET uses the following basic data types in place of the corresponding COM types:

COM Data Type	.NET Data Type
BSTR	string
long	int
double	double
VARIANT_BOOL	bool

14.2.2 .NET Entity Class

Entity is an abstract base class that includes basic functionality used by other Config Store classes. Most of the class is not publically visible, but the class itself is visible and is used in several constructors including the main ConfigStore constructor.

The Entity class defines the following public property:

- Name

The following public Config Store classes are derived from Entity.

- HardwareAsset
- PublishedApi
- SoftwareModule
- PhysicalName
- PhysicalRange
- LogicalName
- Session
- DriverSession
- VirtualName
- VirtualRange
- DataComponent
- IviStructure
- IviInteger
- IviReal
- IviBoolean
- IviString
- IviAPIReference

For additional uses of the Entity class in .NET constructors, refer to 5.3.5.2, *Params Parameters in Constructors*.

14.2.2.1 The Entity.Name property.

The definition of Entity.Name matches the definition of the Name property in the classes that derive from entity.

In cases where a class places additional constriants on the Name property, the validation of Entity.Name is overridden by the class:

- Published Api
- IVI Physical Name
- IVI Virtual Name

14.2.3 .NET Enumerations

The .NET API defines the following enumerations:

- Config Store Location
- IVI Published API Name
- IVI Published API Type
- Session Usage

14.2.3.1Config Store Location

The Config Store Location enumeration provides members for the two standard configuration store XML file locations.

Members do not have explicitly defined values.

Name	***Description***			
		Language	***Identifier***	***Value***
ConfigStoreLocation.Master	The master location.			
		.NET	`ConfigStoreLocation.Master`	
ConfigStoreLocation.ProcessDefault	The default location for the current process.			
		.NET	`ConfigStoreLocation.ProcessDefault`	

14.2.3.2IVI Published API Name

The IVI Published API Name enumeration provides members for each of the standard APIs defined for IVI Instrument Drivers.

Members do not have explicitly defined values.

Name	***Description***			
		Language	***Identifier***	***Value***
IviPublishedApiName.IviDmm	The IviDmm instrument class.			
		.NET	`IviPublishedApiName.IviDmm`	
IviPublishedApiName.IviDriver	The IviDriver inherent capabilities class.			
		.NET	`IviPublishedApiName.IviDriver`	
IviPublishedApiName.IviScope	The IviScope instrument class.			
		.NET	`IviPublishedApiName.IviScope`	
IviPublishedApiName.IviFgen	The IviFgen instrument class.			
		.NET	`IviPublishedApiName.IviFgen`	
IviPublishedApiName.IviDCPwr	The IviDCPwr instrument class.			
		.NET	`IviPublishedApiName.IviDCPwr`	
IviPublishedApiName.IviACPwr	The IviACPwr instrument class.			
		.NET	`IviPublishedApiName.IviACPwr`	

IviPublishedApiName.IviSwtch	The IviSwtch instrument class.			
		.NET	`IviPublishedApiName.IviSwtch`	
IviPublishedApiName.IviPwrMeter	The IviPwrMeter instrument class.			
		.NET	`IviPublishedApiName.IviPwrMeter`	
IviPublishedApiName.IviSpecAn	The IviSpecAn instrument class.			
		.NET	`IviPublishedApiName.IviSpecAn`	
IviPublishedApiName.IviRFSigGen	The IviRFSigGen instrument class.			
		.NET	`IviPublishedApiName.IviRFSigGen`	
IviPublishedApiName.IviCounter	The IviCounter instrument class.			
		.NET	`IviPublishedApiName.IviCounter`	
IviPublishedApiName.IviDownConverter	The IviDownconverter instrument class.			
		.NET	`IviPublishedApiName.IviDownconverter`	
IviPublishedApiName.IviUpConverter	The IviUpconverter instrument class.			
		.NET	`IviPublishedApiName.IviUpconverter`	
IviPublishedApiName.IviDigitizer	The IviDigitizer instrument class.			
		.NET	`IviPublishedApiName.IviDigitizer`	
IviPublishedApiName.IviLxiSync	The IviLxiSync instrument class.			
		.NET	`IviPublishedApiName.IviLxiSync`	

14.2.3.3IVI Published API Type

The IVI Published API Type enumeration provides members for the three standard API types defined for IVI Instrument Drivers.

Members do not have explicitly defined values.

Name	***Description***			
		Language	***Identifier***	***Value***
IviPublishedApiType.IviCom	The IVI-COM API type.			
		.NET	`IviPublishedApiType.IviCom`	
IviPublishedApiType.IviNet	The IVI.NET API type.			
		.NET	`IviPublishedApiType.IviNet`	
IviPublishedApiType.IviC	The IVI-C API type.			
		.NET	`IviPublishedApiType.IviC`	

14.2.3.4Session Usage

The Config Store Location enumeration provides members for the three standard values for data component Used In Session properties.

Name	*Description*			
		Language	*Identifier*	*Value*
SessionUsage.None	A session does not need to define a value for this data component.			
		.NET	`SessionUsage.None`	`0`
SessionUsage. Required	A session must define a value for this data component.			
		.NET	`SessionUsage.Required`	`1`
SessionUsage. Optional	A session may define a value for this data component.			
		.NET	`SessionUsage.Optional`	`2`

14.2.4 Collections in .NET

Refer to section 4.5, *Collections in .NET* for a description of collections in .NET.

14.2.5 .NET Constructors

In contrast to the COM API which does not allow parameterized constructors, the .NET API uses parameterized constructors, so that new objects have legal and consistent data from the start.

14.2.5.1Constructors and Property Access

Each constructor includes parameters for

- Properties that serve as keys in the corresponding collection. For example, Name is the key value for nearly all classes, and so name is a common constructor parameter.
- Properties that should not be changed after they are set initially. For example, the Prefix property for SoftwareModules only needs to be set once, and does not need to be changed after that.

In general, if a constructor includes a parameter, the corresponding property is read only.

14.2.5.2Params Parameters in Constructors

Some constructors include a params parameter. In the .NET API, these allow a calling program to specify members of the object's collections. For example, the constructor for the Virtual Name class includes the `virtualRanges` parameter, which allows the calling program to specify multiple Virtual Range objects to be added to the Virtual Name's Virtual Range collection.

In most cases the type of the params parameter is an array of a base type, either Entity or DataComponent. This allows the calling program to specify any object whose class derives from the base class. The constructor is then responsible for ensuring that the specified objects are valid for that constructor. For example, the Config Store constructor has a params parameter of type Entity[]. This allows the calling program to specify any class that derives from Entity as an argument. However, the Config Store constructor checks each argument to make sure that they are valid for the constructor, and only six types are valid: Driver Sessions, Hardware Assets, Logical Names, Published APIs, Sessions, and Software Modules. If the calling program specifies an object of an invalid derived type, the constructor will throw an exception.

Constructors may enforce additional order constraints on the specified objects. The most common constraint is for items that must be added to one of the six global collections before they are referenced elsewhere. If an order constraint is violated, the constructor will throw an exception.

14.2.5.3Collection Constructors

Collection constructors are not public. Instead, the Configuration Server creates collections when needed, and allows calling programs to populate the collections using the collection Add() methods or constructor params arguments.

14.2.6 .NET Static Methods and Properties

The .NET API includes a few static methods and properties. These are documented for the classes where they are defined, in separate sections to emphasize the fact that they are static. For the most part these methods and properties do not have an exact match in the C or COM APIs.

The most important of the static methods is the factory method Load(…). Load combines instantiating a ConfigStore object and deserializing a configuration store file, returning the instantiated and loaded object. In contract, the ConfigStore constructor should only be used to create a new configuration store from scratch.

14.2.7 .NET Schema Validation

The .NET API provides more control over validation than the C or COM APIs. The default for all of the APIs is to validate an XML file against the Configuration Store XML schema when a file is deserialized. In addition, the .NET API allows a calling program to deserialize a file without validation (for performance) and to validate a file without deserializing it.

14.2.8 .NET Exceptions

Refer to section 5.12.2, *IVI.NET Error Handling*, of *IVI-3.1, Driver Architecture Specification*, for a general overview of exceptions in IVI.NET components. Calling programs should be able to accommodate arbitrary exceptions, including those defined by the .NET Framework.

The .NET Configuration Server does not define any new exceptions. When it explicitly throws an exception, it throws one of the standard .NET exceptions (usually ArgumentException or InvalidOperationException) with a custom message that indicates the problem.

The Configuration Server does not generate warnings.

IVI Configurable Components Class (Virtual)

- Data Components

This section describes each reference.

14.2.9 Hardware Asset

API Technology	Data Type	Access
COM	IIviHardwareAsset**	R/O
C	IviHardwareAssetHandle	R/O
.NET	HardwareAsset	R/O

COM/.NET Property Name

HardwareAsset

C Function Prototype

```
ViStatus _VI_FUNC IviConfig_GetSessionHardwareAssetReference
                                                (IviSessionHandle SessionHandle,
                                                IviHardwareAssetHandle*
                                                HardwareAssetHandle);

ViStatus _VI_FUNC IviConfig_SetSessionHardwareAssetReference
                                                (IviSessionHandle SessionHandle,
                                                IviHardwareAssetHandle
                                                HardwareAssetHandle);
```

C Parameters

Inputs	Description	Datatype
SessionHandle	Handle to an IviSession object.	IviSessionHandle

Outputs	Description	Datatype
HardwareAssetHandle	Handle to an IviHardwareAsset object.	IviHardwareAssetHandle

Description

References the Hardware Asset used by the Session.

Return Values

The *IVI-3.2: Inherent Capabilities Specification* defines general status codes that this function can return.

The table below specifies additional IVI configuration server status codes for this function.

Completion Codes	Description
Not In Global Collection	The item does not exist in the global collection.

14.2.10 Software Module

API Technology	Data Type	Access
COM	IIviSoftwareModule**	R/O
C	IviSoftwareModuleHandle	R/O
.NET	SoftwareModule	R/O

COM/.NET Property Name

SoftwareModule

C Function Prototype

```
ViStatus _VI_FUNC IviConfig_GetSessionSoftwareModuleReference
                                                (IviSessionHandle SessionHandle,
                                                IviSoftwareModuleHandle*
                                                SoftwareModuleHandle);

ViStatus _VI_FUNC IviConfig_SetSessionSoftwareModuleReference
                                                (IviSessionHandle SessionHandle,
                                                IviSoftwareModuleHandle
                                                SoftwareModuleHandle);
```

C Parameters

Inputs	Description	Datatype
SessionHandle	Handle to an IviSession object.	IviSessionHandle

Outputs	Description	Datatype
SoftwareModuleHandle	Handle to an IviSoftwareModule object.	IviSoftwareModuleHandle

Description

References the Software Module configured by the Session.

Return Values

The *IVI-3.2: Inherent Capabilities Specification* defines general status codes that this function can return.

The table below specifies additional IVI configuration server status codes for this function.

Completion Codes	Description
Not In Global Collection	The item does not exist in the global collection.

14.2.11 Virtual Names

API Technology	Data Type	Access
COM	IIviVirtualNameCollection**	R/O
C	IviVirtualNameCollectionHandle	R/O
.NET	VirtualNameCollection	R/O

COM/.NET Property Name

```
VirtualNames
```

C Function Prototype

```
ViStatus _VI_FUNC IviConfig_GetSessionVirtualNameCollection
                                                (IviSessionHandle SessionHandle,
                                                IviVirtualNameCollectionHandle*
                                                VirtualNameCollectionHandle);
```

C Parameters

Inputs	Description	Datatype
SessionHandle	Handle to an IviSession object.	IviSessionHandle

Outputs	Description	Datatype
VirtualNameCollectionHandle	Handle to an IviVirtualNameCollection object.	IviVirtualNameCollectionHandle

Description

References a collection of all the IVI Virtual Name objects used by the Session. Within a collection, the Name property uniquely identifies an IVI Virtual name object. IVI Virtual Name objects cannot be reused between sessions.

14.3 IVI Session Properties

The IVI Session class defines the following property:

- Software Module Name

The IVI Session class inherits the following properties from IVI Configurable Component - Section 5.3, *.NET API Special Features Overview*

This section defines special features of the .NET Configuration Server.

14.3.1 .NET Data Types

.NET uses the following basic data types in place of the corresponding COM types:

COM Data Type	.NET Data Type
BSTR	string
long	int
double	double
VARIANT_BOOL	bool

14.3.2 .NET Entity Class

Entity is an abstract base class that includes basic functionality used by other Config Store classes. Most of the class is not publically visible, but the class itself is visible and is used in several constructors including the main ConfigStore constructor.

The Entity class defines the following public property:

- Name

The following public Config Store classes are derived from Entity.

- HardwareAsset
- PublishedApi
- SoftwareModule
- PhysicalName
- PhysicalRange
- LogicalName
- Session
- DriverSession
- VirtualName
- VirtualRange
- DataComponent
- IviStructure
- IviInteger
- IviReal
- IviBoolean
- IviString
- IviAPIReference

For additional uses of the Entity class in .NET constructors, refer to 5.3.5.2, *Params Parameters in Constructors*.

14.3.2.1 The Entity.Name property.

The definition of Entity.Name matches the definition of the Name property in the classes that derive from entity.

In cases where a class places additional constriants on the Name property, the validation of Entity.Name is overridden by the class:

- Published Api
- IVI Physical Name
- IVI Virtual Name

14.3.3 .NET Enumerations

The .NET API defines the following enumerations:

- Config Store Location
- IVI Published API Name
- IVI Published API Type
- Session Usage

14.3.3.1 Config Store Location

The Config Store Location enumeration provides members for the two standard configuration store XML file locations.

Members do not have explicitly defined values.

Name	***Description***			
		Language	***Identifier***	***Value***
ConfigStoreLocation.Master	The master location.			
		.NET	`ConfigStoreLocation.Master`	
ConfigStoreLocation.ProcessDefault	The default location for the current process.			
		.NET	`ConfigStoreLocation.ProcessDefault`	

14.3.3.2 IVI Published API Name

The IVI Published API Name enumeration provides members for each of the standard APIs defined for IVI Instrument Drivers.

Members do not have explicitly defined values.

Name	***Description***			
		Language	***Identifier***	***Value***
IviPublishedApiName.IviDmm	The IviDmm instrument class.			
		.NET	`IviPublishedApiName.IviDmm`	
IviPublishedApiName.IviDriver	The IviDriver inherent capabilities class.			
		.NET	`IviPublishedApiName.IviDriver`	
IviPublishedApiName.IviScope	The IviScope instrument class.			
		.NET	`IviPublishedApiName.IviScope`	
IviPublishedApiName.IviFgen	The IviFgen instrument class.			
		.NET	`IviPublishedApiName.IviFgen`	
IviPublishedApiName.IviDCPwr	The IviDCPwr instrument class.			
		.NET	`IviPublishedApiName.IviDCPwr`	
IviPublishedApiName.IviACPwr	The IviACPwr instrument class.			
		.NET	`IviPublishedApiName.IviACPwr`	

IviPublishedApiName.IviSwtch	The IviSwtch instrument class.			
		.NET	`IviPublishedApiName.IviSwtch`	
IviPublishedApiName.IviPwrMeter	The IviPwrMeter instrument class.			
		.NET	`IviPublishedApiName.IviPwrMeter`	
IviPublishedApiName.IviSpecAn	The IviSpecAn instrument class.			
		.NET	`IviPublishedApiName.IviSpecAn`	
IviPublishedApiName.IviRFSigGen	The IviRFSigGen instrument class.			
		.NET	`IviPublishedApiName.IviRFSigGen`	
IviPublishedApiName.IviCounter	The IviCounter instrument class.			
		.NET	`IviPublishedApiName.IviCounter`	
IviPublishedApiName.IviDownConverter	The IviDownconverter instrument class.			
		.NET	`IviPublishedApiName.IviDownconverter`	
IviPublishedApiName.IviUpConverter	The IviUpconverter instrument class.			
		.NET	`IviPublishedApiName.IviUpconverter`	
IviPublishedApiName.IviDigitizer	The IviDigitizer instrument class.			
		.NET	`IviPublishedApiName.IviDigitizer`	
IviPublishedApiName.IviLxiSync	The IviLxiSync instrument class.			
		.NET	`IviPublishedApiName.IviLxiSync`	

14.3.3.3IVI Published API Type

The IVI Published API Type enumeration provides members for the three standard API types defined for IVI Instrument Drivers.

Members do not have explicitly defined values.

Name	***Description***			
		Language	***Identifier***	***Value***
IviPublishedApiType.IviCom	The IVI-COM API type.			
		.NET	`IviPublishedApiType.IviCom`	
IviPublishedApiType.IviNet	The IVI.NET API type.			
		.NET	`IviPublishedApiType.IviNet`	
IviPublishedApiType.IviC	The IVI-C API type.			
		.NET	`IviPublishedApiType.IviC`	

14.3.3.4Session Usage

The Config Store Location enumeration provides members for the three standard values for data component Used In Session properties.

Name	*Description*			
		Language	*Identifier*	*Value*
SessionUsage.None	A session does not need to define a value for this data component.			
		.NET	`SessionUsage.None`	`0`
SessionUsage. Required	A session must define a value for this data component.			
		.NET	`SessionUsage.Required`	`1`
SessionUsage. Optional	A session may define a value for this data component.			
		.NET	`SessionUsage.Optional`	`2`

14.3.4 Collections in .NET

Refer to section 4.5, *Collections in .NET* for a description of collections in .NET.

14.3.5 .NET Constructors

In contrast to the COM API which does not allow parameterized constructors, the .NET API uses parameterized constructors, so that new objects have legal and consistent data from the start.

14.3.5.1 Constructors and Property Access

Each constructor includes parameters for

- Properties that serve as keys in the corresponding collection. For example, Name is the key value for nearly all classes, and so name is a common constructor parameter.
- Properties that should not be changed after they are set initially. For example, the Prefix property for SoftwareModules only needs to be set once, and does not need to be changed after that.

In general, if a constructor includes a parameter, the corresponding property is read only.

14.3.5.2 Params Parameters in Constructors

Some constructors include a params parameter. In the .NET API, these allow a calling program to specify members of the object's collections. For example, the constructor for the Virtual Name class includes the `virtualRanges` parameter, which allows the calling program to specify multiple Virtual Range objects to be added to the Virtual Name's Virtual Range collection.

In most cases the type of the params parameter is an array of a base type, either Entity or DataComponent. This allows the calling program to specify any object whose class derives from the base class. The constructor is then responsible for ensuring that the specified objects are valid for that constructor. For example, the Config Store constructor has a params parameter of type Entity[]. This allows the calling program to specify any class that derives from Entity as an argument. However, the Config Store constructor checks each argument to make sure that they are valid for the constructor, and only six types are valid: Driver Sessions, Hardware Assets, Logical Names, Published APIs, Sessions, and Software Modules. If the calling program specifies an object of an invalid derived type, the constructor will throw an exception.

Constructors may enforce additional order constraints on the specified objects. The most common constraint is for items that must be added to one of the six global collections before they are referenced elsewhere. If an order constraint is violated, the constructor will throw an exception.

14.3.5.3Collection Constructors

Collection constructors are not public. Instead, the Configuration Server creates collections when needed, and allows calling programs to populate the collections using the collection Add() methods or constructor params arguments.

14.3.6 .NET Static Methods and Properties

The .NET API includes a few static methods and properties. These are documented for the classes where they are defined, in separate sections to emphasize the fact that they are static. For the most part these methods and properties do not have an exact match in the C or COM APIs.

The most important of the static methods is the factory method Load(…). Load combines instantiating a ConfigStore object and deserializing a configuration store file, returning the instantiated and loaded object. In contract, the ConfigStore constructor should only be used to create a new configuration store from scratch.

14.3.7 .NET Schema Validation

The .NET API provides more control over validation than the C or COM APIs. The default for all of the APIs is to validate an XML file against the Configuration Store XML schema when a file is deserialized. In addition, the .NET API allows a calling program to deserialize a file without validation (for performance) and to validate a file without deserializing it.

14.3.8 .NET Exceptions

Refer to section 5.12.2, *IVI.NET Error Handling*, of *IVI-3.1, Driver Architecture Specification*, for a general overview of exceptions in IVI.NET components. Calling programs should be able to accommodate arbitrary exceptions, including those defined by the .NET Framework.

The .NET Configuration Server does not define any new exceptions. When it explicitly throws an exception, it throws one of the standard .NET exceptions (usually ArgumentException or InvalidOperationException) with a custom message that indicates the problem.

The Configuration Server does not generate warnings.

IVI Configurable Components Class (Virtual)

- Description
- Name

This section describes the behavior and requirements of the property defined in the IVI Session class.

14.3.9 Software Module Name

API Technology	Data Type	Access
.NET	string	R/O
C	ViString	R/O
COM	BSTR	R/O

COM/.NET Property Name

SoftwareModuleName

C Constant Name

IVICONFIG_VAL_SESSION_SOFTWARE_MODULE_NAME

Description

The name of the current or most recently referenced software module referenced by the Software Module property.

14.4 IVI Session Constructor (.NET Only)

The .NET IVI Session class defines one public constructor.

This section describes the behavior and requirements of the constructor.

14.4.1 Session Constructor

Description

Creates an instance of a Session.

Children parameters may be any one of the following classes deriving from Entity:
- Hardware Asset
- Software Module
- Virtual Name
- Data Component

If there are no children parameters, the Session is created without any corresponding references.

Children parameters are added to the corresponding collections. The order does not matter.
- If a Hardware Asset or Software Module object is referenced by the Session, that Hardware Asset or Software Module object must have already been added to the global Hardware Asset Collection or Software Module Collection, respectively.

.NET Prototype

```
public Session(string name, params Entity[] children)
```

.NET Parameters

Inputs	Description	.NET Type
`name`	The name of the Session.	`string`
`children`	Zero or one Hardware Asset, zero or one Software Module, zero to n Virtual Names, and zero to n data components to be referenced by the Session.	`params Entity[]`

15. IVI Driver Session Class

15.1 IVI Driver Session Overview

The IVI Driver Session class inherits from the IVI Session class and adds several properties that may be configured for IVI instrument driver software modules. These properties are common to all IVI instrument drivers and are defined in *IVI-3.2: Inherent Capabilities Specification*.

15.2 IVI Driver Session References

The IVI Driver Session class inherits the following references from IVI Session - Section 13.3.2, Description

API Technology	Data Type	Access
.NET	string	R/W
C	ViString	R/W
COM	BSTR	R/W

COM/.NET Property Name

Description

C Constant Name

IVICONFIG_VAL_CONFIG_COMPONENT_DESCRIPTION

Description

The description of the logical name. The empty string is a legal value for this property.

15.3 IVI Logical Name Constructors (.NET Only)

The .NET IVI Logical Name class defines one public constructor.

This section describes the behavior and requirements of the constructor.

15.3.1 LogicalName Constructors

Description

Creates an instance of a Logical Name.

If a Session object is referenced by the Logical Name, that Session object must have already been added to the global Session Collection.

.NET Prototype

```
public LogicalName(string name, Session session = null)
```

.NET Parameters

Inputs	Description	.NET Type
`name`	The name of the Logical Name.	`string`
`session`	A reference to a Session object that is already a member of the parent ConfigStore's Sessions collection, or null.	`Session`

IVI Session Class:
- Hardware Asset
- Software Module
- Virtual Names

The IVI Driver Session class inherits the following reference from IVI Configurable Component – Section 5.3, *.NET API Special Features Overview*

This section defines special features of the .NET Configuration Server.

15.3.2 .NET Data Types

.NET uses the following basic data types in place of the corresponding COM types:

COM Data Type	.NET Data Type
BSTR	string
long	int
double	double
VARIANT_BOOL	bool

15.3.3 .NET Entity Class

Entity is an abstract base class that includes basic functionality used by other Config Store classes. Most of the class is not publically visible, but the class itself is visible and is used in several constructors including the main ConfigStore constructor.

The Entity class defines the following public property:
- Name

The following public Config Store classes are derived from Entity.
- HardwareAsset
- PublishedApi
- SoftwareModule
- PhysicalName
- PhysicalRange
- LogicalName
- Session
- DriverSession
- VirtualName
- VirtualRange
- DataComponent
- IviStructure
- IviInteger
- IviReal
- IviBoolean
- IviString
- IviAPIReference

For additional uses of the Entity class in .NET constructors, refer to 5.3.5.2, *Params Parameters in Constructors*.

15.3.3.1 The Entity.Name property.

The definition of Entity.Name matches the definition of the Name property in the classes that derive from entity.

In cases where a class places additional constriants on the Name property, the validation of Entity.Name is overridden by the class:

- Published Api
- IVI Physical Name
- IVI Virtual Name

15.3.4 .NET Enumerations

The .NET API defines the following enumerations:

- Config Store Location
- IVI Published API Name
- IVI Published API Type
- Session Usage

15.3.4.1Config Store Location

The Config Store Location enumeration provides members for the two standard configuration store XML file locations.

Members do not have explicitly defined values.

Name	***Description***			
		Language	***Identifier***	***Value***
ConfigStoreLocation.Master	The master location.			
		.NET	`ConfigStoreLocation.Master`	
ConfigStoreLocation.ProcessDefault	The default location for the current process.			
		.NET	`ConfigStoreLocation.ProcessDefault`	

15.3.4.2IVI Published API Name

The IVI Published API Name enumeration provides members for each of the standard APIs defined for IVI Instrument Drivers.

Members do not have explicitly defined values.

Name	***Description***			
		Language	***Identifier***	***Value***
IviPublishedApiName.IviDmm	The IviDmm instrument class.			
		.NET	`IviPublishedApiName.IviDmm`	
IviPublishedApiName.IviDriver	The IviDriver inherent capabilities class.			
		.NET	`IviPublishedApiName.IviDriver`	
IviPublishedApiName.IviScope	The IviScope instrument class.			
		.NET	`IviPublishedApiName.IviScope`	
IviPublishedApiName.IviFgen	The IviFgen instrument class.			
		.NET	`IviPublishedApiName.IviFgen`	
IviPublishedApiName.IviDCPwr	The IviDCPwr instrument class.			
		.NET	`IviPublishedApiName.IviDCPwr`	
IviPublishedApiName.IviACPwr	The IviACPwr instrument class.			
		.NET	`IviPublishedApiName.IviACPwr`	

IviPublishedApiName.IviSwtch	The IviSwtch instrument class.			
		.NET	`IviPublishedApiName.IviSwtch`	
IviPublishedApiName.IviPwrMeter	The IviPwrMeter instrument class.			
		.NET	`IviPublishedApiName.IviPwrMeter`	
IviPublishedApiName.IviSpecAn	The IviSpecAn instrument class.			
		.NET	`IviPublishedApiName.IviSpecAn`	
IviPublishedApiName.IviRFSigGen	The IviRFSigGen instrument class.			
		.NET	`IviPublishedApiName.IviRFSigGen`	
IviPublishedApiName.IviCounter	The IviCounter instrument class.			
		.NET	`IviPublishedApiName.IviCounter`	
IviPublishedApiName.IviDownConverter	The IviDownconverter instrument class.			
		.NET	`IviPublishedApiName.IviDownconverter`	
IviPublishedApiName.IviUpConverter	The IviUpconverter instrument class.			
		.NET	`IviPublishedApiName.IviUpconverter`	
IviPublishedApiName.IviDigitizer	The IviDigitizer instrument class.			
		.NET	`IviPublishedApiName.IviDigitizer`	
IviPublishedApiName.IviLxiSync	The IviLxiSync instrument class.			
		.NET	`IviPublishedApiName.IviLxiSync`	

15.3.4.3IVI Published API Type

The IVI Published API Type enumeration provides members for the three standard API types defined for IVI Instrument Drivers.

Members do not have explicitly defined values.

Name	***Description***			
		Language	***Identifier***	***Value***
IviPublishedApiType.IviCom	The IVI-COM API type.			
		.NET	`IviPublishedApiType.IviCom`	
IviPublishedApiType.IviNet	The IVI.NET API type.			
		.NET	`IviPublishedApiType.IviNet`	
IviPublishedApiType.IviC	The IVI-C API type.			
		.NET	`IviPublishedApiType.IviC`	

15.3.4.4Session Usage

The Config Store Location enumeration provides members for the three standard values for data component Used In Session properties.

Name	*Description*			
		Language	*Identifier*	*Value*
SessionUsage.None	A session does not need to define a value for this data component.			
		.NET	`SessionUsage.None`	`0`
SessionUsage. Required	A session must define a value for this data component.			
		.NET	`SessionUsage.Required`	`1`
SessionUsage. Optional	A session may define a value for this data component.			
		.NET	`SessionUsage.Optional`	`2`

15.3.5 Collections in .NET

Refer to section 4.5, *Collections in .NET* for a description of collections in .NET.

15.3.6 .NET Constructors

In contrast to the COM API which does not allow parameterized constructors, the .NET API uses parameterized constructors, so that new objects have legal and consistent data from the start.

15.3.6.1 Constructors and Property Access

Each constructor includes parameters for

- Properties that serve as keys in the corresponding collection. For example, Name is the key value for nearly all classes, and so name is a common constructor parameter.
- Properties that should not be changed after they are set initially. For example, the Prefix property for SoftwareModules only needs to be set once, and does not need to be changed after that.

In general, if a constructor includes a parameter, the corresponding property is read only.

15.3.6.2 Params Parameters in Constructors

Some constructors include a params parameter. In the .NET API, these allow a calling program to specify members of the object's collections. For example, the constructor for the Virtual Name class includes the `virtualRanges` parameter, which allows the calling program to specify multiple Virtual Range objects to be added to the Virtual Name's Virtual Range collection.

In most cases the type of the params parameter is an array of a base type, either Entity or DataComponent. This allows the calling program to specify any object whose class derives from the base class. The constructor is then responsible for ensuring that the specified objects are valid for that constructor. For example, the Config Store constructor has a params parameter of type Entity[]. This allows the calling program to specify any class that derives from Entity as an argument. However, the Config Store constructor checks each argument to make sure that they are valid for the constructor, and only six types are valid: Driver Sessions, Hardware Assets, Logical Names, Published APIs, Sessions, and Software Modules. If the calling program specifies an object of an invalid derived type, the constructor will throw an exception.

Constructors may enforce additional order constraints on the specified objects. The most common constraint is for items that must be added to one of the six global collections before they are referenced elsewhere. If an order constraint is violated, the constructor will throw an exception.

15.3.6.3 Collection Constructors

Collection constructors are not public. Instead, the Configuration Server creates collections when needed, and allows calling programs to populate the collections using the collection Add() methods or constructor params arguments.

15.3.7 .NET Static Methods and Properties

The .NET API includes a few static methods and properties. These are documented for the classes where they are defined, in separate sections to emphasize the fact that they are static. For the most part these methods and properties do not have an exact match in the C or COM APIs.

The most important of the static methods is the factory method Load(…). Load combines instantiating a ConfigStore object and deserializing a configuration store file, returning the instantiated and loaded object. In contract, the ConfigStore constructor should only be used to create a new configuration store from scratch.

15.3.8 .NET Schema Validation

The .NET API provides more control over validation than the C or COM APIs. The default for all of the APIs is to validate an XML file against the Configuration Store XML schema when a file is deserialized. In addition, the .NET API allows a calling program to deserialize a file without validation (for performance) and to validate a file without deserializing it.

15.3.9 .NET Exceptions

Refer to section 5.12.2, *IVI.NET Error Handling*, of *IVI-3.1, Driver Architecture Specification*, for a general overview of exceptions in IVI.NET components. Calling programs should be able to accommodate arbitrary exceptions, including those defined by the .NET Framework.

The .NET Configuration Server does not define any new exceptions. When it explicitly throws an exception, it throws one of the standard .NET exceptions (usually ArgumentException or InvalidOperationException) with a custom message that indicates the problem.

The Configuration Server does not generate warnings.

IVI Configurable Components Class (Virtual):
- Data Components

15.4 IVI Driver Session Properties

The IVI Driver Session class defines the following properties:
- Cache
- Driver Setup
- Interchange Check
- Query Instrument Status
- Range Check
- Record Coercions
- Simulate

The IVI Driver Session class inherits the following properties from IVI Configurable Component – Section 5.3, *.NET API Special Features Overview*

This section defines special features of the .NET Configuration Server.

15.4.1 .NET Data Types

.NET uses the following basic data types in place of the corresponding COM types:

COM Data Type	.NET Data Type
BSTR	string
long	int
double	double
VARIANT_BOOL	bool

15.4.2 .NET Entity Class

Entity is an abstract base class that includes basic functionality used by other Config Store classes. Most of the class is not publically visible, but the class itself is visible and is used in several constructors including the main ConfigStore constructor.

The Entity class defines the following public property:

- Name

The following public Config Store classes are derived from Entity.

- HardwareAsset
- PublishedApi
- SoftwareModule
- PhysicalName
- PhysicalRange
- LogicalName
- Session
- DriverSession
- VirtualName
- VirtualRange
- DataComponent
- IviStructure
- IviInteger
- IviReal
- IviBoolean
- IviString
- IviAPIReference

For additional uses of the Entity class in .NET constructors, refer to 5.3.5.2, *Params Parameters in Constructors*.

15.4.2.1 The Entity.Name property.

The definition of Entity.Name matches the definition of the Name property in the classes that derive from entity.

In cases where a class places additional constriants on the Name property, the validation of Entity.Name is overridden by the class:

- Published Api
- IVI Physical Name
- IVI Virtual Name

15.4.3 .NET Enumerations

The .NET API defines the following enumerations:

- Config Store Location
- IVI Published API Name
- IVI Published API Type
- Session Usage

15.4.3.1Config Store Location

The Config Store Location enumeration provides members for the two standard configuration store XML file locations.

Members do not have explicitly defined values.

Name	***Description***			
		Language	***Identifier***	***Value***
ConfigStoreLocation.Master	The master location.			
		.NET	`ConfigStoreLocation.Master`	
ConfigStoreLocation.ProcessDefault	The default location for the current process.			
		.NET	`ConfigStoreLocation.ProcessDefault`	

15.4.3.2IVI Published API Name

The IVI Published API Name enumeration provides members for each of the standard APIs defined for IVI Instrument Drivers.

Members do not have explicitly defined values.

Name	***Description***			
		Language	***Identifier***	***Value***
IviPublishedApiName.IviDmm	The IviDmm instrument class.			
		.NET	`IviPublishedApiName.IviDmm`	
IviPublishedApiName.IviDriver	The IviDriver inherent capabilities class.			
		.NET	`IviPublishedApiName.IviDriver`	
IviPublishedApiName.IviScope	The IviScope instrument class.			
		.NET	`IviPublishedApiName.IviScope`	
IviPublishedApiName.IviFgen	The IviFgen instrument class.			
		.NET	`IviPublishedApiName.IviFgen`	
IviPublishedApiName.IviDCPwr	The IviDCPwr instrument class.			
		.NET	`IviPublishedApiName.IviDCPwr`	
IviPublishedApiName.IviACPwr	The IviACPwr instrument class.			
		.NET	`IviPublishedApiName.IviACPwr`	

IviPublishedApiName.IviSwtch	The IviSwtch instrument class.			
		.NET	`IviPublishedApiName.IviSwtch`	
IviPublishedApiName.IviPwrMeter	The IviPwrMeter instrument class.			
		.NET	`IviPublishedApiName.IviPwrMeter`	
IviPublishedApiName.IviSpecAn	The IviSpecAn instrument class.			
		.NET	`IviPublishedApiName.IviSpecAn`	
IviPublishedApiName.IviRFSigGen	The IviRFSigGen instrument class.			
		.NET	`IviPublishedApiName.IviRFSigGen`	
IviPublishedApiName.IviCounter	The IviCounter instrument class.			
		.NET	`IviPublishedApiName.IviCounter`	
IviPublishedApiName.IviDownConverter	The IviDownconverter instrument class.			
		.NET	`IviPublishedApiName.IviDownconverter`	
IviPublishedApiName.IviUpConverter	The IviUpconverter instrument class.			
		.NET	`IviPublishedApiName.IviUpconverter`	
IviPublishedApiName.IviDigitizer	The IviDigitizer instrument class.			
		.NET	`IviPublishedApiName.IviDigitizer`	
IviPublishedApiName.IviLxiSync	The IviLxiSync instrument class.			
		.NET	`IviPublishedApiName.IviLxiSync`	

15.4.3.3IVI Published API Type

The IVI Published API Type enumeration provides members for the three standard API types defined for IVI Instrument Drivers.

Members do not have explicitly defined values.

Name	***Description***			
		Language	***Identifier***	***Value***
IviPublishedApiType.IviCom	The IVI-COM API type.			
		.NET	`IviPublishedApiType.IviCom`	
IviPublishedApiType.IviNet	The IVI.NET API type.			
		.NET	`IviPublishedApiType.IviNet`	
IviPublishedApiType.IviC	The IVI-C API type.			
		.NET	`IviPublishedApiType.IviC`	

15.4.3.4Session Usage

The Config Store Location enumeration provides members for the three standard values for data component Used In Session properties.

Name	*Description*			
		Language	*Identifier*	*Value*
SessionUsage.None	A session does not need to define a value for this data component.			
		.NET	`SessionUsage.None`	`0`
SessionUsage. Required	A session must define a value for this data component.			
		.NET	`SessionUsage.Required`	`1`
SessionUsage. Optional	A session may define a value for this data component.			
		.NET	`SessionUsage.Optional`	`2`

15.4.4 Collections in .NET

Refer to section 4.5, *Collections in .NET* for a description of collections in .NET.

15.4.5 .NET Constructors

In contrast to the COM API which does not allow parameterized constructors, the .NET API uses parameterized constructors, so that new objects have legal and consistent data from the start.

15.4.5.1Constructors and Property Access

Each constructor includes parameters for

- Properties that serve as keys in the corresponding collection. For example, Name is the key value for nearly all classes, and so name is a common constructor parameter.
- Properties that should not be changed after they are set initially. For example, the Prefix property for SoftwareModules only needs to be set once, and does not need to be changed after that.

In general, if a constructor includes a parameter, the corresponding property is read only.

15.4.5.2Params Parameters in Constructors

Some constructors include a params parameter. In the .NET API, these allow a calling program to specify members of the object's collections. For example, the constructor for the Virtual Name class includes the `virtualRanges` parameter, which allows the calling program to specify multiple Virtual Range objects to be added to the Virtual Name's Virtual Range collection.

In most cases the type of the params parameter is an array of a base type, either Entity or DataComponent. This allows the calling program to specify any object whose class derives from the base class. The constructor is then responsible for ensuring that the specified objects are valid for that constructor. For example, the Config Store constructor has a params parameter of type Entity[]. This allows the calling program to specify any class that derives from Entity as an argument. However, the Config Store constructor checks each argument to make sure that they are valid for the constructor, and only six types are valid: Driver Sessions, Hardware Assets, Logical Names, Published APIs, Sessions, and Software Modules. If the calling program specifies an object of an invalid derived type, the constructor will throw an exception.

Constructors may enforce additional order constraints on the specified objects. The most common constraint is for items that must be added to one of the six global collections before they are referenced elsewhere. If an order constraint is violated, the constructor will throw an exception.

15.4.5.3Collection Constructors

Collection constructors are not public. Instead, the Configuration Server creates collections when needed, and allows calling programs to populate the collections using the collection Add() methods or constructor params arguments.

15.4.6 .NET Static Methods and Properties

The .NET API includes a few static methods and properties. These are documented for the classes where they are defined, in separate sections to emphasize the fact that they are static. For the most part these methods and properties do not have an exact match in the C or COM APIs.

The most important of the static methods is the factory method Load(…). Load combines instantiating a ConfigStore object and deserializing a configuration store file, returning the instantiated and loaded object. In contract, the ConfigStore constructor should only be used to create a new configuration store from scratch.

15.4.7 .NET Schema Validation

The .NET API provides more control over validation than the C or COM APIs. The default for all of the APIs is to validate an XML file against the Configuration Store XML schema when a file is deserialized. In addition, the .NET API allows a calling program to deserialize a file without validation (for performance) and to validate a file without deserializing it.

15.4.8 .NET Exceptions

Refer to section 5.12.2, *IVI.NET Error Handling*, of *IVI-3.1, Driver Architecture Specification*, for a general overview of exceptions in IVI.NET components. Calling programs should be able to accommodate arbitrary exceptions, including those defined by the .NET Framework.

The .NET Configuration Server does not define any new exceptions. When it explicitly throws an exception, it throws one of the standard .NET exceptions (usually ArgumentException or InvalidOperationException) with a custom message that indicates the problem.

The Configuration Server does not generate warnings.

IVI Configurable Components Class (Virtual):

- Description
- Name

This section describes the behavior and requirements of each property defined in the IviDriver Session class.

15.4.9 Cache

API Technology	Data Type	Access
.NET	bool	R/W
C	ViBoolean	R/W
COM	VARIANT_BOOL	R/W

COM/.NET Property Name

Cache

C Constant Name

IVICONFIG_VAL_DRIVER_SESSION_CACHE

Description

Cache stores the Boolean value that initializes the Cache attribute in an IVI instrument specific driver. For a complete description of the Cache attribute, refer to Section 5.1, *Cache*, of *IVI-3.2: Inherent Capabilities Specification*.

15.4.10 Driver Setup

API Technology	Data Type	Access
.NET	string	R/W
C	ViString	R/W
COM	BSTR	R/W

COM/.NET Property Name

DriverSetup

C Constant Name

IVICONFIG_VAL_DRIVER_SESSION_DRIVER_SETUP

Description

Driver Setup stores the string that initializes the Driver Setup attribute in an IVI instrument specific driver. For a complete description of the Driver Setup attribute, refer to Section 5.16, *Driver Setup*, of *IVI-3.2: Inherent Capabilities Specification*.

The empty string is a legal value for this property.

15.4.11 Interchange Check

API Technology	Data Type	Access
.NET	bool	R/W
C	ViBoolean	R/W
COM	VARIANT_BOOL	R/W

COM/.NET Property Name

InterchangeCheck

C Constant Name

IVICONFIG_VAL_DRIVER_SESSION_INTERCHANGE_CHECK

Description

Interchange Check stores the Boolean value that initializes the Interchange Check attribute in an IVI instrument specific driver. For a complete description of the Interchange Check attribute, refer to Section 5.22, *Interchange Check*, of *IVI-3.2: Inherent Capabilities Specification*.

15.4.12 Query Instrument Status

API Technology	Data Type	Access
.NET	bool	R/W
C	ViBoolean	R/W
COM	VARIANT_BOOL	R/W

COM/.NET Property Name

QueryInstrStatus

C Constant Name

IVICONFIG_VAL_DRIVER_SESSION_QUERY_INSTR_STATUS

Description

Query Instrument Status stores the Boolean value that initializes the Query Instrument Status attribute in an IVI instrument specific driver. For a complete description of the Query Instrument Status attribute, refer to Section 5.24, *Query Instrument Status*, of *IVI-3.2: Inherent Capabilities Specification*.

15.4.13 Range Check

API Technology	Data Type	Access
.NET	bool	R/W
C	ViBoolean	R/W
COM	VARIANT_BOOL	R/W

COM/.NET Property Name

RangeCheck

C Constant Name

IVICONFIG_VAL_DRIVER_SESSION_RANGE_CHECK

Description

Range Check stores the Boolean value that initializes the Range Check attribute in an IVI instrument specific driver. For a complete description of the Range Check attribute, refer to Section 5.25, *Range Check*, of *IVI-3.2: Inherent Capabilities Specification*.

15.4.14 Record Value Coercions

API Technology	Data Type	Access
.NET	bool	R/W
C	ViBoolean	R/W
COM	VARIANT_BOOL	R/W

COM/.NET Property Name

RecordCoercions

C Constant Name

IVICONFIG_VAL_DRIVER_SESSION_RECORD_COERCIONS

Description

Record Value Coercions stores the Boolean value that initializes the Record Value Coercions attribute in an IVI instrument specific driver. For a complete description of the Record Value Coercions attribute, refer to Section 5.26, *Record Value Coercions*, of *IVI-3.2: Inherent Capabilities Specification*.

15.4.15 Simulate

API Technology	Data Type	Access
.NET	bool	R/W
C	ViBoolean	R/W
COM	VARIANT_BOOL	R/W

COM/.NET Property Name

```
Simulate
```

C Constant Name

```
IVICONFIG_VAL_DRIVER_SESSION_SIMULATE
```

Description

Simulate stores the Boolean value that initializes the Simulate attribute in an IVI instrument specific driver. For a complete description of the Simulate attribute, refer to Section 5.27, *Simulate*, of *IVI-3.2: Inherent Capabilities Specification*.

15.5 IVI Driver Session Constructor (.NET Only)

The .NET IVI Driver Session class defines one public constructor.

This section describes the behavior and requirements of the constructor.

15.5.1 DriverSession Constructor

Description

Creates an instance of a Driver Session.

Children parameters may be any one of the following classes deriving from Entity:
- Hardware Asset
- Software Module
- Virtual Name
- Data Component

If there are no children parameters, the Driver Session is created without any corresponding references.

Children parameters are added to the corresponding collections. The order does not matter.
- If a Hardware Asset or Software Module object is referenced by the Session, that Hardware Asset or Software Module object must have already been added to the global Hardware Asset Collection or Software Module Collection, respectively.

.NET Prototype

```
public DriverSession(string name, params Entity[] children)
```

.NET Parameters

Inputs	Description	.NET Type
`name`	The name of the Driver Session.	`string`
`children`	Zero or one Hardware Asset, zero or one Software Module, zero to n Virtual Names, and zero to n data components to be referenced by the Session.	`params Entity[]`

16. IVI Virtual Name Class

16.1 IVI Virtual Name Overview

IVI Virtual Name objects describe virtual identifiers defined by the user for a particular session. Virtual identifiers are the names that a client program gives to instances of the repeated capabilities it accesses in software modules. For example, an IVI Scope specific instrument driver might name channels “A”, “B”, and “C” while the client programmer might choose to name them “Chan1”, “Chan2”, and “Chan3”. In this case, “A”, “B”, and “C” are physical identifiers defined by the software module and “Chan1”, “Chan2”, and “Chan3” are the corresponding virtual identifiers used by the client program.

Name is the virtual identifier string. It may be qualified by referenced Virtual Ranges.

MapTo is the string that is substituted for the virtual identifier. It may be qualified by referenced Virtual Ranges.

An IVI Virtual Name object can reference a collection of IVI Virtual Range objects, which allows an efficient way of creating a large number of virtual identifiers by appending integers from the range to the Name property.

For a comprehensive overview of the treatment of repeated capabilities in the Configuration Server, including the role that IVI Virtual Names play, refer to Section 2.9, *Repeated Capabilities*.

16.2 IVI Virtual Name Reference

The IVI Virtual Name class defines the following reference:

- Virtual Ranges

This section describes the reference.

16.2.1 Virtual Ranges

API Technology	Data Type	Access
COM	IIviVirtualRangeCollection**	R/O
C	IviVirtualRangeCollectionHandle	R/O
.NET	VirtualRangeCollection	R/O

COM/.NET Property Name

VirtualRanges

C Function Prototype

```
ViStatus _VI_FUNC IviConfig_GetVirtualNameVirtualRangeCollection
                                                (IviVirtualNameHandle
                                                VirtualNameHandle,
                                                IviVirtualRangeCollectionHandle*
                                                VirtualRangeCollectionHandle);
```

C Parameters

Inputs	Description	Datatype
VirtualNameHandle	Handle to an IviVirtualName object.	IviVirtualNameHandle

Outputs	Description	Datatype
VirtualRangeCollectionHandle	Handle to an IviVirtualRangeCollection object.	IviVirtualRangeCollectionHandle

Description

References a collection of IVI Virtual Range objects used to qualify the referencing IVI Virtual Name object.

16.3 IVI Virtual Name Properties

The IVI Virtual Name class defines the following properties:

- Map To
- Name

This section describes the behavior and requirements of each property.

16.3.1 Map To

API Technology	Data Type	Access
.NET	string	R/W
C	ViString	R/W
COM	BSTR	R/W

COM/.NET Property Name

MapTo

C Constant Name

IVICONFIG_VAL_VIRTUAL_NAME_MAPTO

Description

A string used to replace the virtual identifier when the virtual identifier is used to specify repeated capability instances in a software module. When the software module encounters the associated virtual identifier in a repeated capability selector, it substitutes the Map To string for the virtual identifier. The software module is responsible for determining if the resulting string is a valid physical repeated capability selector in context. Refer to section 5.9.2, *Applying Virtual Identifier Mappings*, of *IVI-3.1: Driver Architecture Specification* for a description of how software modules resolve virtual identifiers.

The empty string is a legal value for this property only if the IVI Virtual Name object references a non-empty collection of IVI Virtual Range objects.

16.3.2 Name

API Technology	Data Type	Access
.NET	string	R/O
C	ViString	R/W
COM	BSTR	R/W

COM/.NET Property Name

Name

C Constant Name

IVICONFIG_VAL_VIRTUAL_NAME_NAME

Description

The virtual identifier name. When the software module encounters the virtual identifier name in a repeated capability selector, it substitutes the Map To string for the virtual identifier. The software module is responsible for determining if the resulting string is a valid physical repeated capability selector in context. Refer to section 5.9.2, *Applying Virtual Identifier Mappings*, of *IVI-3.1: Driver Architecture Specification* for a description of how software modules resolve virtual identifiers. Name shall consist of one or more of the following characters: ‘a’-‘z’, ‘A’-‘Z’, ‘0’-‘9’, ‘!’, and ‘_’.

The empty string is not a legal value for this property.

16.4 IVI Virtual Name Constructors (.NET Only)

The .NET IVI Virtual Name class defines one public constructor.

This section describes the behavior and requirements of the constructor.

16.4.1 VirtualName Constructors

Description

Creates an instance of a Virtual Name.

If there are no virtualRanges parameters, the Virtual Name is created without any Virtual Range references.

VirtualRanges parameters are added to the corresponding collection. The order does not matter.

.NET Prototype

```
public VirtualName(string name,
                   string mapTo,
                   params VirtualRange[] virtualRanges)
```

.NET Parameters

Inputs	Description	.NET Type
`name`	The name of the Virtual Name.	`string`
`mapTo`	The string that is substituted by the software module for Name whenever it is encountered in a repeated capability selector. The empty string is a legal value for this property only if the virtual range collection is non-empty.	`string`
`virtualRanges`	Zero to n Virtual Ranges to be referenced by the Virtual Name	`params VirtualRange []`

17. IVI Virtual Range Class

17.1 IVI Virtual Range Overview

The IVI Virtual Range class allows multiple virtual identifiers to be defined with a minimum of effort. An IVI Virtual Range object shall be referenced by exactly one IVI Virtual Name object.

For a comprehensive overview of the treatment of repeated capabilities in the Configuration Server, including the role that virtual names play, refer to Section 2.9, *Repeated Capabilities*.

17.2 IVI Virtual Range Properties

The IVI Virtual Range class defines the following properties:

- Max
- Min
- Name
- Starting Physical Index

This section describes the behavior and requirements of each property.

17.2.1 Max

API Technology	Data Type	Access
.NET	int	R/W
C	ViInt32	R/W
COM	long	R/W

COM/.NET Property Name

Max

C Constant Name

IVICONFIG_VAL_VIRTUAL_RANGE_MAX

Description

The upper bound of a range of integers to be appended to the Name property of the referencing IVI Virtual Name object to create a set of virtual repeated capability identifiers.

17.2.2 Min

API Technology	Data Type	Access
.NET	int	R/W
C	ViInt32	R/W
COM	long	R/W

COM/.NET Property Name

Min

C Constant Name

IVICONFIG_VAL_VIRTUAL_RANGE_MIN

Description

The lower bound of a range of integers to be appended to the Name property of the referencing IVI Virtual Name object to create a set of virtual repeated capability identifiers.

17.2.3 Name

API Technology	Data Type	Access
.NET	string	R/O
C	ViString	R/W
COM	BSTR	R/W

COM/.NET Property Name

Name

C Constant Name

IVICONFIG_VAL_VIRTUAL_RANGE_NAME

Description

The name of the IVI Virtual Range. This name is used to uniquely identify the range in the collection, and is not used in creating the set of virtual identifiers.

The empty string is not a legal value for this property.

17.2.4 Starting Physical Index

API Technology	Data Type	Access
.NET	int	R/W
C	ViInt32	R/W
COM	long	R/W

COM/.NET Property Name

StartingPhysicalIndex

C Constant Name

IVICONFIG_VAL_VIRTUAL_RANGE_START_PHYSICAL_INDEX

Description

When a range of integers is appended to the Name property of the referencing IVI Virtual Name object to create a set of virtual repeated capability identifiers, Starting Physical Index is added to each integer in the range, and the result is appended to the Map To property to obtain the corresponding set of physical identifiers.

17.3 IVI Virtual Range Constructor (.NET Only)

The .NET IVI Virtual Range class defines one public constructor.

This section describes the behavior and requirements of the constructor.

17.3.1 VirtualRange Constructor

Description

Creates an instance of a Virtual Range.

.NET Prototype

```
public VirtualRange(string name, int min, int max, int startingPhysicalIndex)
```

.NET Parameters

Inputs	Description	.NET Type
`name`	The name of the Virtual Range.	`string`
`min`	The lower bound of a range of integers to be appended to the Name property of the referencing IVI Virtual Name object to create a set of physical repeated capability identifiers.	`int`
`max`	The upper bound of a range of integers to be appended to the Name property of the referencing IVI Virtual Name object to create a set of physical repeated capability identifiers.	`int`
`startingPhysicalIndex`	When a range of integers is appended to the Name property of the referencing IVI VirtualName object to create a set of virtual repeated capability identifiers, startingPhysicalIndex is added to each integer in the range, and the result is appended to the VirtualName.MapTo property to obtain the corresponding set of physical identifiers.	`int`

18. IVI Data Component Class

18.1 IVI Data Component Overview

The IVI Data Component class is used to extend the set of properties that can be supported by other Configuration Server classes. This class is not implemented directly – it is a virtual base class. The following Configuration Server classes inherit from the IVI Data Component class:

- IVI API Reference
- IVI Boolean
- IVI Integer
- IVI Real
- IVI String
- IVI Structure

Each of these classes inherits Name, Description, ReadOnly, Used In Session, and Type properties from IVI Data Component class.

All IVI Configuration Server classes that inherit from the IVI Configurable Component class include a reference to a collection of IVI Data Components. In addition, the IVI Structure class also contains a reference to a collection of Data Components, allowing the hierarchical definition of a data components structure.

18.2 IVI Data Component Properties

The IVI Data Component class defines the following properties:

- Description
- Help Context ID
- Help File Path
- Name
- Read Only
- Software Module Key
- Type
- Used In Session

This section describes the behavior and requirements of each property.

18.2.1 Description

API Technology	Data Type	Access
.NET	string	R/W
C	ViString	R/W
COM	BSTR	R/W

COM/.NET Property Name

Description

C Constant Name

IVICONFIG_VAL_DATA_COMPONENT_DESCRIPTION

Description

A description of the IVI Data Component.

The empty string is a legal value for this property.

18.2.2 Help Context ID

API Technology	Data Type	Access
.NET	int	R/W
C	ViInt32	R/W
COM	long	R/W

COM/.NET Property Name

HelpContextID

C Constant Name

IVICONFIG_VAL_DATA_COMPONENT_HELP_CONTEXT_ID

Description

The help context ID for the data component.

The default value for this property shall be 0. The value of this property is not meaningful if the Help File property does not contain a valid help file pathname. If the Help File property does contain a valid help file pathname, Help Context ID shall return the help context ID for the topic in the help file that provides the help for the data component.

The configuration server does not verify the content of this property. The person or tool that maintains the field is responsible for its validity.

18.2.3 Help File Path

API Technology	Data Type	Access
.NET	string	R/W
C	ViString	R/W
COM	BSTR	R/W

COM/.NET Property Name

HelpFilePath

C Constant Name

IVICONFIG_VAL_DATA_COMPONENT_HELP_FILE_PATH

Description

The fully qualified pathname of the help file that provides the help for the data component.

The default value for this property shall be the empty string. If Help File Path is not the empty string, it shall contain the fully qualified pathname of a help file that conforms to one of the standard formats for Windows help. If Help File Path is not the empty string, Help Context ID shall return the help context ID for the topic in the help file that provides the help for the data component.

The configuration server does not verify the content of this property. The person or tool that maintains the field is responsible for its validity.

18.2.4 Name

API Technology	Data Type	Access
.NET	string	R/O
C	ViString	R/W
COM	BSTR	R/W

COM/.NET Property Name

Name

C Constant Name

IVICONFIG_VAL_DATA_COMPONENT_NAME

Description

The name of the IVI Data Component.

The empty string is not a legal value for this property.

18.2.5 Read Only

API Technology	Data Type	Access
.NET	bool	R/W
C	ViBoolean	R/W
COM	VARIANT_BOOL	R/W

COM/.NET Property Name

ReadOnly

C Constant Name

IVICONFIG_VAL_DATA_COMPONENT_READ_ONLY

Description

Indicates whether the value of the data component may be changed in a configuration store GUI. If True, a Configuration Utility must not allow the user to modify the contents of the data component object. If False, the associated data component objects may be read or written.

18.2.6 Software Module Key

API Technology	Data Type	Access
.NET	string	R/W
C	ViString	R/W
COM	BSTR	R/W

COM/.NET Property Name

SoftwareModuleKey

C Constant Name

IVICONFIG_VAL_DATA_COMPONENT_SW_MODULE_KEY

Description

A string that the software module uses to identify the internal software element to which the data component applies.

18.2.7 Type

API Technology	Data Type	Access
.NET	string	R/O
C	ViString	R/O
COM	BSTR	R/O

COM/.NET Property Name

Type

C Constant Name

IVICONFIG_VAL_DATA_COMPONENT_TYPE

Description

A string denoting the data type of the associated data element. Valid values for this field are “Structure”, “Integer”, “Real”, “Boolean”, “String”, and “APIReference”.

18.2.8 Used In Session

API Technology	Data Type	Access
.NET	SessionUsage	R/W
C	ViString	R/W
COM	BSTR	R/W

COM/.NET Property Name

UsedInSession

C Constant Name

IVICONFIG_VAL_DATA_COMPONENT_USED_IN_SESSION

Description

This property determines whether a data component associated with a software module is used in a session that references the software module. The possible values are “Required”, “Optional”, and “None”. The values are case insensitive.

For data components associated with software modules and sessions, Used In Session shall be “Required”, “Optional”, or “None”. For data components associated with hardware assets, Used In Session shall be “None”.

If the value is “Required”, the data component must be present as part of a session’s data components, with a valid value, for the associated software module to operate correctly.

If the value is “Optional”, the data component may be present as part of a session’s data components, but is not required for the associated software module to operate correctly. This allows the user to choose whether to override the driver’s default value.

If the value is “None”, the data component is ignored by the software module. Its presence and value have no effect on the operation of the software module.

The configuration server shall enforce the valid values for this field. If an invalid value is entered manually, the configuration server shall return an Invalid Value error.

18.3 IVI Data Component Constructors (.NET Only)

The Ivi Data Component class is a base class that is never constructed directly. Only the derived classes have constructors.

19. IVI Structure Class

19.1 IVI Structure Overview

The IVI Structure class allows one collection of IVI Data Components to reference another collection of IVI Data Components. This allows data components to be structured hierarchically.

19.2 IVI Structure References

The IVI Structure class defines the following reference:

- DataComponents

This section describes the reference.

19.2.1 Data Components

API Technology	Data Type	Access
COM	IIviDataComponentCollection**	R/O
C	IviDataComponentCollectionHandle	R/O
.NET	DataComponentCollection	R/O

COM/.NET Property Name

DataComponents

C Function Prototype

```
ViStatus _VI_FUNC IviConfig_GetStructureDataComponentCollection
                                                (IviDataComponentHandle
                                                StructureHandle,
                                                IviDataComponentCollectionHandle*
                                                DataComponentCollectionHandle);
```

C Parameters

Inputs	Description	Datatype
StructureHandle	Handle to an IviDataComponent object.	IviDataComponentHandle

Outputs	Description	Datatype
DataComponentCollectionHandle	Handle to an IviDataComponentCollection object.	IviDataComponentCollectionHandle

Description

References a collection of IVI Data Component objects. Circular references or a circular series of references are not allowed.

19.3 IVI Structure Properties

The following properties are inherited from IVI Data Component - Section 0,

IVI Virtual Range Constructor (.NET Only)

The .NET IVI Virtual Range class defines one public constructor.

This section describes the behavior and requirements of the constructor.

19.3.1 VirtualRange Constructor

Description

Creates an instance of a Virtual Range.

.NET Prototype

```
public VirtualRange(string name, int min, int max, int startingPhysicalIndex)
```

.NET Parameters

Inputs	Description	.NET Type
`name`	The name of the Virtual Range.	`string`
`min`	The lower bound of a range of integers to be appended to the Name property of the referencing IVI Virtual Name object to create a set of physical repeated capability identifiers.	`int`
`max`	The upper bound of a range of integers to be appended to the Name property of the referencing IVI Virtual Name object to create a set of physical repeated capability identifiers.	`int`
`startingPhysicalIndex`	When a range of integers is appended to the Name property of the referencing IVI VirtualName object to create a set of virtual repeated capability identifiers, startingPhysicalIndex is added to each integer in the range, and the result is appended to the VirtualName.MapTo property to obtain the corresponding set of physical identifiers.	`int`

IVI Data Component Class

- Description
- Help Context ID
- Help File Path
- Name
- Read Only
- Software Module Key
- Type
- Used In Session

In an IVI Structure object, Type shall be set to “Structure”.

19.4 IVI Structure Constructors (.NET Only)

The .NET IVI Structure class defines two public constructors.

This section describes the behavior and requirements of each constructor.

19.4.1 IviStructure Constructors

Description

Creates an instance of the IVI Structure class.

DataComponents parameters may be any one of the following classes deriving from DataComponent:

- IviStructure,
- IviInteger
- IviReal
- IviBoolean
- IviString
- IviAPIReference

DataComponents parameters are added to the IVI Structure's Data Components Collection in the order in which they are specified.

The IVI Published Api referenced by a DataComponents IviAPIReference parameter must have been added to the global Published API Collection before the IVI Structure is added to a global Configurable Component Collection. (Global Configurable Components are Hardware Assets, Software Modules, Sessions, and Driver Sessions.)

.NET Prototype

```
public IviStructure(
            string name,
            params DataComponent[] dataComponents)

public IviStructure(
            string name,
            bool readOnly = true,
            SessionUsage usedInSession = SessionUsage.Optional,
            params DataComponent[] dataComponents)
```

.NET Parameters

Inputs	Description	.NET Type
`name`	The name of the Hardware Asset. This name must be unique in any collection of Hardware Assets which includes this one.	`string`
`readOnly`	If false, the value of the data component may be changed in a configuration store GUI.	`bool`
`usedInSession`	Determines whether a data component associated with a software module is used in a session that references the software module.	`SessionUsage`
`dataComponents`	Zero to n Data Components to be referenced by this Hardware Asset.	`DataComponent[]`

20. IVI Integer Class

20.1 IVI Integer Overview

The IVI Integer class defines a 32-bit integer data type for use in the IVI configuration server. This includes an integer value and other type information. An IVI Integer object cannot exist independently of an IVI Data Components collection.

20.2 IVI Integer Properties

The IVI Integer class defines the following properties:

- Units
- Value

The following properties are inherited from IVI Data Component - Section 0,

20.3 IVI Virtual Range Constructor (.NET Only)

The .NET IVI Virtual Range class defines one public constructor.

This section describes the behavior and requirements of the constructor.

20.3.1 VirtualRange Constructor

Description

Creates an instance of a Virtual Range.

.NET Prototype

```
public VirtualRange(string name, int min, int max, int startingPhysicalIndex)
```

.NET Parameters

Inputs	Description	.NET Type
`name`	The name of the Virtual Range.	`string`
`min`	The lower bound of a range of integers to be appended to the Name property of the referencing IVI Virtual Name object to create a set of physical repeated capability identifiers.	`int`
`max`	The upper bound of a range of integers to be appended to the Name property of the referencing IVI Virtual Name object to create a set of physical repeated capability identifiers.	`int`
`startingPhysicalIndex`	When a range of integers is appended to the Name property of the referencing IVI VirtualName object to create a set of virtual repeated capability identifiers, startingPhysicalIndex is added to each integer in the range, and the result is appended to the VirtualName.MapTo property to obtain the corresponding set of physical identifiers.	`int`

IVI Data Component Class

- Description
- Help Context ID
- Help File Path
- Name
- Read Only
- Software Module Key
- Type
- Used In Session

In an IVI Integer object, type shall be set to “Integer”.

This section describes the behavior and requirements of each property defined in the IVI Integer class.

20.3.2 Units

API Technology	Data Type	Access
.NET	string	R/W
C	ViString	R/W
COM	BSTR	R/W

COM/.NET Property Name

Units

C Constant Name

IVICONFIG_VAL_DATA_COMPONENT_UNITS

Description

A string that specifies the units to be applied to Value.

The empty string is a legal value for this property.

20.3.3 Value

API Technology	Data Type	Access
.NET	int	R/W
C	ViInt32	R/W
COM	long	R/W

COM/.NET Property Name

Value

C Constant Name

IVICONFIG_VAL_DATA_COMPONENT_VALUE

Description

The integer value of the data component.

20.4 IVI Integer Constructor (.NET Only)

The .NET IVI Integer class defines one public constructor.

This section describes the behavior and requirements of the constructor.

20.4.1 IviInteger Constructor

Description

Creates an instance of the IVI Integer class.

.NET Prototype

```
public IviInteger(
            string name,
            long value,
            string units = "",
            bool readOnly = true,
            SessionUsage usedInSession = SessionUsage.Optional)
```

.NET Parameters

Inputs	Description	.NET Type
`name`	The name of the IVI Integer. This name must be unique in any collection of Data Components.	`string`
`value`	The integer value.	`long`
`units`	The units associated with the integer value.	`string`
`readOnly`	If false, the value of the data component may be changed in a configuration store GUI.	`bool`
`usedInSession`	Determines whether a data component associated with a software module is used in a session that references the software module.	`SessionUsage`

21. IVI Real Class

21.1 IVI Real Overview

The IVI Real class defines a 64-bit real data type for use in the IVI configuration server. This includes a real value and other type information. An IVI Real object cannot exist independently of an IVI Data Components collection.

21.2 IVI Real Properties

The IVI Real class defines the following properties:

- Units
- Value

The following properties are inherited from IVI Data Component - Section 0,

21.3 IVI Virtual Range Constructor (.NET Only)

The .NET IVI Virtual Range class defines one public constructor.

This section describes the behavior and requirements of the constructor.

21.3.1 VirtualRange Constructor

Description

Creates an instance of a Virtual Range.

.NET Prototype

```
public VirtualRange(string name, int min, int max, int startingPhysicalIndex)
```

.NET Parameters

Inputs	Description	.NET Type
`name`	The name of the Virtual Range.	`string`
`min`	The lower bound of a range of integers to be appended to the Name property of the referencing IVI Virtual Name object to create a set of physical repeated capability identifiers.	`int`
`max`	The upper bound of a range of integers to be appended to the Name property of the referencing IVI Virtual Name object to create a set of physical repeated capability identifiers.	`int`
`startingPhysicalIndex`	When a range of integers is appended to the Name property of the referencing IVI VirtualName object to create a set of virtual repeated capability identifiers, startingPhysicalIndex is added to each integer in the range, and the result is appended to the VirtualName.MapTo property to obtain the corresponding set of physical identifiers.	`int`

IVI Data Component Class

- Description
- Help Context ID
- Help File Path
- Name
- Read Only
- Software Module Key
- Type
- Used In Session

In an IVI Real object, type shall be set to “Real”.

This section describes the behavior and requirements of each property defined in the IVI Real class.

21.3.2 Units

API Technology	Data Type	Access
.NET	string	R/W
C	ViString	R/W
COM	BSTR	R/W

COM/.NET Property Name

Units

C Constant Name

IVICONFIG_VAL_DATA_COMPONENT_UNITS

Description

A string that specifies the units to be applied to Value.

The empty string is a legal value for this property.

21.3.3 Value

API Technology	Data Type	Access
.NET	double	R/W
C	ViReal64	R/W
COM	double	R/W

COM/.NET Property Name

Value

C Constant Name

IVICONFIG_VAL_DATA_COMPONENT_VALUE

Description

The real value of the data component.

21.4 IVI Real Constructor (.NET Only)

The .NET IVI Real class defines one public constructor.

This section describes the behavior and requirements of the constructor.

21.4.1 IviReal Constructor

Description

Creates an instance of the IVI Real class.

.NET Prototype

```
public IviReal(
            string name,
            double value,
            string units = "",
            bool readOnly = true,
            SessionUsage usedInSession = SessionUsage.Optional)
```

.NET Parameters

Inputs	Description	.NET Type
`name`	The name of the IVI Real. This name must be unique in any collection of Data Components.	`string`
`value`	The real value.	`double`
`units`	The units associated with the real value.	`string`
`readOnly`	If false, the value of the data component may be changed in a configuration store GUI.	`bool`
`usedInSession`	Determines whether a data component associated with a software module is used in a session that references the software module.	`SessionUsage`

22. IVI Boolean Class

22.1 IVI Boolean Overview

The IVI Boolean class defines a Boolean data type for use in the IVI configuration server. This includes a Boolean value and other type information. An IVI Boolean object cannot exist independently of an IVI Data Components collection.

22.2 IVI Boolean Properties

The IVI Boolean class defines the following properties:

- Value

The following properties are inherited from IVI Data Component - Section 0,

22.3 IVI Virtual Range Constructor (.NET Only)

The .NET IVI Virtual Range class defines one public constructor.

This section describes the behavior and requirements of the constructor.

22.3.1 VirtualRange Constructor

Description

Creates an instance of a Virtual Range.

.NET Prototype

```
public VirtualRange(string name, int min, int max, int startingPhysicalIndex)
```

.NET Parameters

Inputs	Description	.NET Type
`name`	The name of the Virtual Range.	`string`
`min`	The lower bound of a range of integers to be appended to the Name property of the referencing IVI Virtual Name object to create a set of physical repeated capability identifiers.	`int`
`max`	The upper bound of a range of integers to be appended to the Name property of the referencing IVI Virtual Name object to create a set of physical repeated capability identifiers.	`int`
`startingPhysicalIndex`	When a range of integers is appended to the Name property of the referencing IVI VirtualName object to create a set of virtual repeated capability identifiers, startingPhysicalIndex is added to each integer in the range, and the result is appended to the VirtualName.MapTo property to obtain the corresponding set of physical identifiers.	`int`

IVI Data Component Class

- Description
- Help Context ID
- Help File Path
- Name
- Read Only
- Software Module Key
- Type
- Used In Session

In an IVI Boolean object, type shall be set to “Boolean”.

This section describes the behavior and requirements of the property defined in the IVI Boolean class.

22.3.2 Value

API Technology	Data Type	Access
.NET	bool	R/W
C	ViBoolean	R/W
COM	VARIANT_BOOL	R/W

COM/.NET Property Name

Value

C Constant Name

IVICONFIG_VAL_DATA_COMPONENT_VALUE

Description

The Boolean value of the data component.

22.4 IVI Boolean Constructors (.NET Only)

The .NET IVI Boolean class defines one public constructor.

This section describes the behavior and requirements of the constructor.

22.4.1 IviBoolean Constructor

Description

Creates an instance of the IVI Boolean class.

.NET Prototype

```
public IviBoolean(
            string name,
            bool value,
            bool readOnly = true,
            SessionUsage usedInSession = SessionUsage.Optional)
```

.NET Parameters

Inputs	Description	.NET Type
`name`	The name of the IVI Boolean. This name must be unique in any collection of Data Components.	`string`
`value`	The Boolean value.	`bool`
`readOnly`	If false, the value of the data component may be changed in a configuration store GUI.	`bool`
`usedInSession`	Determines whether a data component associated with a software module is used in a session that references the software module.	`SessionUsage`

23. IVI String Class

23.1 IVI String Overview

The IVI String class defines a string data type for use in the IVI configuration server. This includes a string value and other type information. An IVI String object cannot exist independently of an IVI Data Components collection.

23.2 IVI String Properties

The IVI String class defines the following properties:

- Value

The following properties are inherited from IVI Data Component - Section 0,

23.3 IVI Virtual Range Constructor (.NET Only)

The .NET IVI Virtual Range class defines one public constructor.

This section describes the behavior and requirements of the constructor.

23.3.1 VirtualRange Constructor

Description

Creates an instance of a Virtual Range.

.NET Prototype

```
public VirtualRange(string name, int min, int max, int startingPhysicalIndex)
```

.NET Parameters

Inputs	Description	.NET Type
`name`	The name of the Virtual Range.	`string`
`min`	The lower bound of a range of integers to be appended to the Name property of the referencing IVI Virtual Name object to create a set of physical repeated capability identifiers.	`int`
`max`	The upper bound of a range of integers to be appended to the Name property of the referencing IVI Virtual Name object to create a set of physical repeated capability identifiers.	`int`
`startingPhysicalIndex`	When a range of integers is appended to the Name property of the referencing IVI VirtualName object to create a set of virtual repeated capability identifiers, startingPhysicalIndex is added to each integer in the range, and the result is appended to the VirtualName.MapTo property to obtain the corresponding set of physical identifiers.	`int`

IVI Data Component Class

- Description
- Help Context ID
- Help File Path
- Name
- Read Only
- Software Module Key
- Type
- Used In Session

In an IVI String object, Type shall be set to “String”.

This section describes the behavior and requirements of the property defined in the IVI String class.

23.3.2 Value

API Technology	Data Type	Access
.NET	string	R/W
C	ViString	R/W
COM	BSTR	R/W

COM/.NET Property Name

Value

C Constant Name

IVICONFIG_VAL_DATA_COMPONENT_VALUE

Description

The string value of the data component.

The empty string is a legal value for this property.

23.4 IVI String Constructors (.NET Only)

The .NET IVI String class defines one public constructor.

This section describes the behavior and requirements of the constructor.

23.4.1 IviString Constructor

Description

Creates an instance of the IVI String class.

.NET Prototype

```
public IviString(
            string name,
            string value,
            bool readOnly = true,
            SessionUsage usedInSession = SessionUsage.Optional)
```

.NET Parameters

Inputs	Description	.NET Type
`name`	The name of the IVI String. This name must be unique in any collection of Data Components.	`string`
`value`	The string value.	`string`
`readOnly`	If false, the value of the data component may be changed in a configuration store GUI.	`bool`
`usedInSession`	Determines whether a data component associated with a software module is used in a session that references the software module.	`SessionUsage`

24. IVI API Reference Class

24.1 IVI API Reference Overview

The IVI API Reference class defines a reference to a Published API data type for use in the Configuration Server. An IVI API Reference object cannot exist independently of an IVI Data Components collection. All API references are configurable initial settings for either a software module or a session.

An IVI Software Module uses IVI API Reference objects to describe the published APIs that it uses (not the ones that it implements). A software module (the "using" software module) "uses" an API when it makes API calls to another software module (the "used" software module) that implements the API. When an IVI API Reference object is created at install time in a software module's data components collection, the value of the Published API reference is set and may not be subsequently changed.

When an IVI Session includes an IVI API Reference object, it associates the Published API with a session or driver session. This session configures the "used" software module chosen to implement the published API. The Value property is a session name or logical name. Get Session and Get Driver Session may be used to resolve the name to a session. Refer to Section 7.4.3, *Get Session* and Section 7.4.2, *Get Driver Session* for more details.

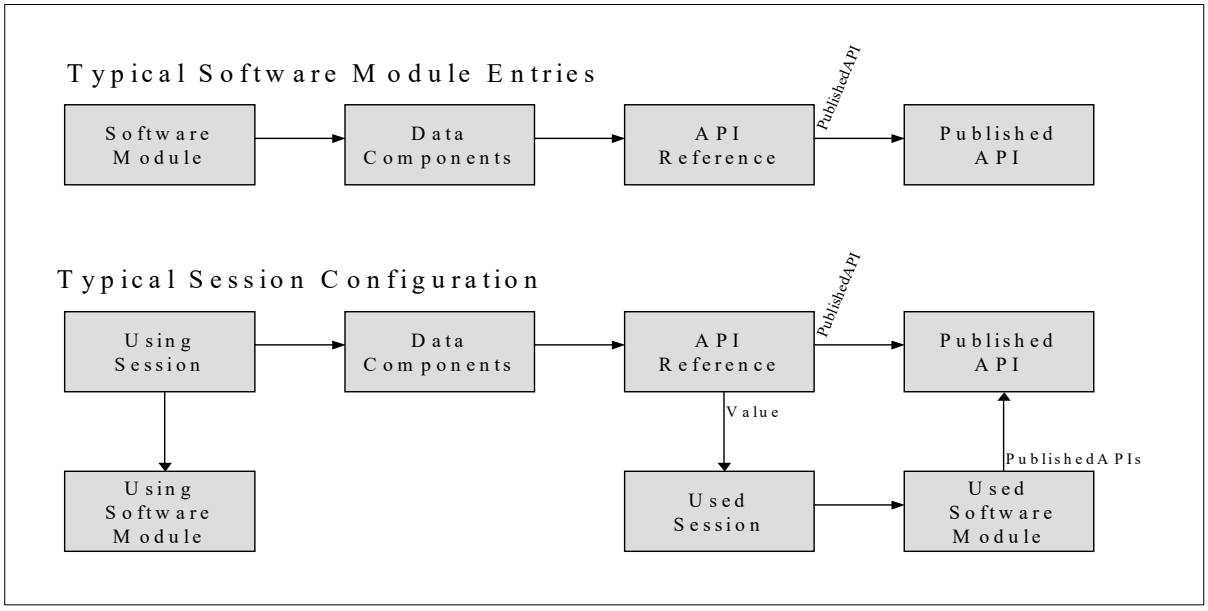

Figure 24-1 Typical API Reference Configuration Store Entries

Figure 24-1 shows two typical uses of the API reference data component. The first shows the set of configuration store entries for a software module that uses an API reference. The second shows the set of configuration store entries for a session that configures the software module.

While an API Reference may be used by IVI Drivers, the typical use is for IVI-MSS components.

Attempts to add an IVI API Reference object will fail with an Invalid Data Component error if one of the following conditions is true:

- The data components collection to which is being added is referenced by an IVI Hardware Asset object.
- Its Used In Session property is "None".

24.2 IVI API Reference References

The IVI API Reference class defines the following reference:

- Published API

This section describes the reference.

24.2.1 Published API

API Technology	Data Type	Access
COM	IIviPublishedsAPI**	R/O
C	IviPublishedAPIsHandle	R/O
.NET	PublishedAPI	R/O

COM/.NET Property Name

PublishedAPI

C Function Prototype

```
ViStatus _VI_FUNC IviConfig_GetAPIReferencePublishedAPIReference
                                                (IviDataComponentHandle
                                                ApiReferenceHandle,
                                                IviPublishedAPIsHandle*
                                                PublishedAPIsHandle);

ViStatus _VI_FUNC IviConfig_SetAPIReferencePublishedAPIReference
                                                (IviDataComponentHandle
                                                ApiReferenceHandle,
                                                IviPublishedAPIsHandle
                                                PublishedAPIsHandle);
```

C Parameters

Inputs	Description	Datatype
APIReferenceHandle	Handle to an IviDataComponent object.	IviDataComponentHandle

Outputs	Description	Datatype
PublishedAPIHandle	Handle to an IviPublishedAPI object.	IviPublishedAPIsHandle

Description

The Published API property references a published API to be associated with the client role identified by the Name property. The Value property designates a session name that implements the published API.

24.3 IVI API Reference Properties

The IVI API Reference class defines the following properties:

- Value

The following properties are inherited from IVI Data Component - Section 0,

24.4 IVI Virtual Range Constructor (.NET Only)

The .NET IVI Virtual Range class defines one public constructor.

This section describes the behavior and requirements of the constructor.

24.4.1 VirtualRange Constructor

Description

Creates an instance of a Virtual Range.

.NET Prototype

```
public VirtualRange(string name, int min, int max, int startingPhysicalIndex)
```

.NET Parameters

Inputs	Description	.NET Type
`name`	The name of the Virtual Range.	`string`
`min`	The lower bound of a range of integers to be appended to the Name property of the referencing IVI Virtual Name object to create a set of physical repeated capability identifiers.	`int`
`max`	The upper bound of a range of integers to be appended to the Name property of the referencing IVI Virtual Name object to create a set of physical repeated capability identifiers.	`int`
`startingPhysicalIndex`	When a range of integers is appended to the Name property of the referencing IVI VirtualName object to create a set of virtual repeated capability identifiers, startingPhysicalIndex is added to each integer in the range, and the result is appended to the VirtualName.MapTo property to obtain the corresponding set of physical identifiers.	`int`

IVI Data Component Class

- Description
- Help Context ID
- Help File Path
- Name
- Read Only
- Software Module Key
- Type
- Used In Session

In an IVI API Reference object, Type shall be set to “APIReference”.

This section describes the behavior and requirements of the property defined in the IVI API Reference class.

24.4.2 Value

API Technology	Data Type	Access
.NET	string	R/W
C	ViString	R/W
COM	BSTR	R/W

COM/.NET Property Name

Value

C Constant Name

IVICONFIG_VAL_DATA_COMPONENT_VALUE

Description

A logical name or session name. Value can be passed to GetSession() or GetDrverSession() in the Name parameter. A session reference is returned according to the semantics defined for GetSession() and GetDriverSession().

The empty string is a legal value for this property.

24.5 IVI API Reference Constructor (.NET Only)

The .NET IVI API Reference class defines one public constructor.

This section describes the behavior and requirements of the constructor.

24.5.1 IviAPIReference Constructor

Description

Creates an instance of an IVI API Reference.

Children parameters are added to the corresponding collections in the order in which they are specified.

Before a Published API object may be added to an IVI API Reference, that Published API object must have already been added to the global Published APIs Collection.

.NET Prototype

```
public IviAPIReference(
            string name,
            string value,
            PublishedApi publishedApi,
            bool readOnly = true,
            SessionUsage usedInSession = SessionUsage.Optional
```

.NET Parameters

Inputs	Description	.NET Type
`name`	The name of the IVI String. This name must be unique in any collection of Data Components.	`string`
`value`	A logical name, session name, or driver session name.	`string`
`publishedApi`	A reference to a single PublishedAPI object that is already a member of the parent ConfigStore's PublishedAPIs collection.	`PublishedApi`
`readOnly`	If false, the value of the data component may be changed in a configuration store GUI.	`bool`
`usedInSession`	Determines whether a data component associated with a software module is used in a session that references the software module.	`SessionUsage`

25. Configuration Server Error and Completion Codes

The Configuration Server specification defines the following error codes in addition to the generic IVI error codes defined in *IVI-3.2: Inherent Capabilities*.

Table 25-1 Configuration Server Completion Codes

Error Name	*Description*			
	Language		*Identifier*	*Value(hex)*
Deserialize Failed	The specified configuration store file could not be deserialized.			
	C	IVICONFIG_ERROR_DESERIALIZE_FAILED		0xBFFA1200
	COM	E_IVICONFIG_DESERIALIZE_FAILED		0x80041200
Already Deserialized	A deserialize was attempted after a previous deserialize had already succeeded.			
	C	IVICONFIG_ERROR_ALREADY_DESERIALIZED		0xBFFA1201
	COM	E_IVICONFIG_ALREADY_DESERIALIZED		0x80041201
Serialize Failed	The specified configuration store file could not be serialized.			
	C	IVICONFIG_ERROR_SERIALIZE_FAILED		0xBFFA1202
	COM	E_IVICONFIG_SERIALIZE_FAILED		0x80041202
Session Not Found	The session name or logical name could not be resolved to a session or driver session.			
	C	IVICONFIG_ERROR_SESSION_NOT_FOUND		0xBFFA1203
	COM	E_IVICONFIG_SESSION_NOT_FOUND		0x80041203
Not In Global Collection	The item does not exist in the global collection.			
	C	IVICONFIG_ERROR_NOT_IN_GLOBAL		0xBFFA1204
	COM	E_IVICONFIG_NOT_IN_GLOBAL		0x80041204
Duplicate Entry	An entry with name already exists in the collection.			
	C	IVICONFIG_ERROR_ALREADY_EXIST		0xBFFA1205
	COM	E_IVICONFIG_ALREADY_EXIST		0x80041205
Master Not Found	The registry entry for the master configuration store does not exist or the file could not be found.			
	C	IVICONFIG_ERROR_MASTER_NOT_FOUND		0xBFFA1206
	COM	E_IVICONFIG_MASTER_NOT_FOUND		0x80041206
Does Not Exist	The item does not exist in the collection.			
	C	IVICONFIG_ERROR_NOT_EXIST		0xBFFA1207
	COM	E_IVICONFIG_NOT_EXIST		0x80041207
Invalid Data Component	The data component is not a valid data component.			
	C	IVICONFIG_ERROR_INVALID_DATA_COMPONENT		0xBFFA1208
	COM	E_IVICONFIG_INVALID_DATA_COMPONENT		0x80041208
Invalid Handle	The specified handle is invalid or of an incorrect type.			
	C	IVICONFIG_ERROR_INVALID_HANDLE		0xBFFA1220

		COM	N/A	
Invalid Property ID	The specified property ID is not valid for this function.			
		C	IVICONFIG_ERROR_INVALID_PROPERTY_ID	0xBFFA1221
		COM	N/A	
Reference Still Exists	The element cannot be removed from the global collection when it is referenced in the local collections.			
		C	IVICONFIG_ERROR_LOCAL_REFERENCE_EXIST	0xBFFA1209
		COM	E_IVICONFIG_LOCAL_REFERENCE_EXIST	0x80041209
Not Supported	The operation is not supported.			
		C		0xBFFA1222
		COM		0x80041222
Master Store Registry Conflict	The locations of the master configuration store in the 32-bit and 64-bit registry hives are not the same.			
		C	IVICONFIG_ERROR_MASTER_REGISTRY_CONFLICT	0xBFFA1223
		COM	E_IVICONFIG_MASTER_REGISTRY_CONFLICT	0x80041223

Table 25-2 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.

Table 25-2. Configuration Server Error Message Strings

Name	*Message String*
Deserialize Failed	"IviConfigServer.IviConfigStore.1: Deserialize failed. %1"
Already Deserialized	"IviConfigServer.IviConfigStore.1: A previous deserialize has already succeeded."
Serialize Failed	"IviConfigServer.IviConfigStore.1: Serialize failed. %1"
Session Not Found	"IviConfigServer.IviConfigStore.1: Get%1 failed. Name %2 could not be resolved to a %1."
Not In Global Collection	"IviConfigServer.IviConfigStore.1: %1 failed. %2 does not exist in the global collection or the object is not the same as in the global collection."
Duplicate Entry	"IviConfigServer.IviConfigStore.1: %1 failed. %2 already exists in the collection."
Master Not Found	"IviConfigServer.IviConfigStore.1: get_MasterLocation failed. The registry key does not exist or the file can not be found."
Does Not Exist	"IviConfigServer.IviConfigStore.1: %1 failed. %2 does not exist in the collection."
Invalid Data Component	"IviConfigServer.IviConfigStore.1: : %1 failed. The data component is not a valid data component. %2"
Invalid Handle	"IviConfigServer: %1: The specified handle is either invalid or is of an incorrect type."
Invalid Property ID	"IviConfigStore: %1: The specified property ID is not a valid ID for this function."
Reference Still Exists	"IviConfigStore: %1: %2 failed. The element cannot be removed from the global collection when it is referenced in the local

Name	*Message String*
	collections.”
Not Supported	The operation is not supported.
Master Store Registry Conflict	“IviConfigServer.IviConfigStore.1: get_MasterLocation failed. The locations of the master configuration store in the 32-bit and 64-bit registry hives are not the same.”

26. Configuration Store Data Format

Configuration Store data is stored in an XML file. The format is specified by a schema file, IviConfigurationStore_1-6.xsd which is installed in the same directory as the master configuration store file.

27. Configuration Utility Implementation Guidelines

Configuration utilities facilitate the process of "manually" editing of the configuration store. They are not specified by the IVI Foundation, and so discussion of implementation is confined to these guidelines.

Users will prefer using a configuration utility to either manually editing the configuration store XML file using a text editor, or using an IVI configuration server API to edit the configuration store as needed, since this requires programming. To provide this ease of use, configuration utilities may use a graphical user interface designed to perform the tasks that users must perform to review configuration store content and configure IVI instrument drivers and/or IVI-MSS role control modules. Vendors may choose to tailor the functionality of a configuration utility to particular business needs, or to develop a broadly applicable utility that performs many general configuration server tasks.

Configuration utilities should make it clear to the user that virtual names should be sufficiently specific to the application that they are unlikely to conflict with physical names.

27.1 General

Configuration utilities should always use an IVI configuration server to make any modifications to any configuration store.

Configuration utilities should preserve the data integrity of the configuration store as described in this specification. While the configuration server API handles quite a bit of the data integrity as described in this specification, some items must be handled by the configuration utility and those are described in the following guidelines.

27.2 Hardware Assets

The configuration utility should allow users to add, modify, and delete hardware assets.

The configuration utility should not limit the format of the IO Resource Descriptor so as to limit future potential formats for I/O addresses.

27.3 Published APIs

The installation of a Software Module may add a Published API to the global collection. Configuration utilities may also add Published APIs, but they should be very careful about modifying or deleting them.

27.4 Software Modules

Configuration utilities should not add or delete software modules to the master configuration store. Software modules are added to the configuration store when they are installed, and are deleted when they are uninstalled.

Configuration utilities should not modify any of the data referenced directly or indirectly by software modules. The only exception is that they may add, modify, and delete documentation data components referenced by the software module.

Configuration utilities may copy software modules to, and delete software modules from "slave" configuration stores. When they do, all of the referenced data should also be copied or deleted.

27.5 Sessions

The configuration utility should allow users to add, modify, and delete sessions.

Configuration utilities should be able to configure the configurable initial settings of a session accurately. That means modifying the Value property. Users should not be able to modify the following properties in a configurable initial setting: Description, Help Context ID, Help File Path, Name, Read Only, Software Module Key, Type, Used In Session or Units.

27.6 Documentation Data Components

Configuration utilities should add, modify, or delete documentation data components for a hardware asset, software module, or session accurately. In general, users may modify any property of documentation data components.

28. Limitations

28.1 Distributed Systems

Remote access to the Configuration Server has not been validated to work. Specifying this support will introduce new issues related to DCOM security and system configuration.

28.2 Concurrent Reading and Writing

The Configuration Server does not support multiple concurrent writers or concurrent readers and writers. It does support multiple concurrent readers.

Appendix A: COM Configuration Server API and IVI-COM Driver Example

This example is written using the COM Configuration Server.

The subject of this example is an imaginary IVI-COM instrument specific driver. The driver supports a family of oscilloscopes from GizmoTronics , Inc. The driver supports the IVI inherent interfaces and the IVI scope class-compliant interfaces. In addition to the standard IVI configuration properties, the driver can be configured from the configuration server to turn tracing on and off. The driver supports four channels that it knows as “C1” through “C4”.

All examples are in Visual Basic, and may be abbreviated to emphasize configuration server use.

The configuration server code that needs to be run when the driver is installed looks like this.

```
Option Explicit

Private Sub AddSoftwareModule()

Dim cs As New IviConfigStore
Dim sm As IviSoftwareModule
Dim pa As IviPublishedAPI
Dim pn As IviPhysicalName
Dim pr As IviPhysicalRange
Dim dcb As IviBoolean

'// Deserialize the master configuration store.

On Error GoTo DeserializeError
cs.Deserialize cs.MasterLocation
On Error GoTo 0

'// Delete the old version of the driver

On Error Resume Next
cs.SoftwareModules.Remove "gt40xx"
On Error GoTo 0

'// Make sure that the Published API entries used by the software module
'//  exist in the global Published API collection.

Set pa = New IviPublishedAPI
pa.Name = "IviDriver"
pa.Type = "IVI-COM"
pa.MajorVersion = 2
pa.MinorVersion = 0
     '// If the API is already in the collection, what follows will return
     '//  an error, but it doesn't need to be trapped because the API
     '//  exists in the collection, which is what we want.
On Error Resume Next
cs.PublishedAPIs.Add pa
On Error GoTo 0

Set pa = New IviPublishedAPI
pa.Name = "IviScope"
pa.Type = "IVI-COM"
pa.MajorVersion = 2
pa.MinorVersion = 0
     '// If the API is already in the collection, what follows will return
```

```
            '//  an error, but it doesn't need to be trapped because the API
            '//  exists in the collection, which is what we want.
        On Error Resume Next
        cs.PublishedAPIs.Add pa
        On Error GoTo 0

        '// Create the new software module entry

        Set sm = New IviSoftwareModule
        sm.Name = "gt40xx"
        sm.Description = "IVI-COM Specific Instrument Driver for GT40xx family of
        oscilloscopes"
        sm.Prefix = "gt40xx"
        sm.ProgId = "gt40xx.gt40xx"
        sm.ModulePath = ""
        sm.SupportedInstrumentModels = "gt4000,gt4001,gt4010,gt4011,gt4012"

        '// Add the Published API entries to the software module

        sm.PublishedAPIs.Add cs.PublishedAPIs.Item("IviDriver", 2, 0, "IVI-COM")
        sm.PublishedAPIs.Add cs.PublishedAPIs.Item("IviScope", 2, 0, "IVI-COM")

        '// Add the physical name and physical range entries

        Set pn = New IviPhysicalName
        pn.Name = "C"
        pn.RCName = "Channel"
        sm.PhysicalNames.Add pn

        Set pr = New IviPhysicalRange
        pr.Name = "C Range 1"
        pr.Max = 4
        pr.Min = 1
        pn.PhysicalRanges.Add pr

        '// Add the data components

        Set dcb = New IviBoolean
        dcb.Name = "Trace"
        dcb.Description = "If True, tracing is on, otherwise off"
        '// dcb.Type automatically set to "Boolean" by the API
        dcb.ReadOnly = True
        dcb.UsedInSession = "Required"  '// Software module will default to False
        dcb.Value = False               '// False is the default configuration value
        sm.DataComponents.Add dcb

        cs.SoftwareModules.Add sm


        Exit Sub

        DeserializeError:
        '// Handle the error appropriately.
End Sub
```

Now create a session for the driver. A logical name (“Bob”) will refer to the session. The session will refer to a hardware asset whose resource descriptor is “GPIB0::12::INSTR”. It will also provide logical names for the software modules physical names and configure the values of the trace data component.

The configuration server code that needs to be run when a session is created for the driver software module follows. Bear in mind that most end-users will use a GUI to edit the configuration store, but some users may choose to write code like this – for example, as part of a test system. In any case, this example code is

meant to illustrate the kinds of configuration server entries that must be made. It is not meant to be bulletproof copy and paste code.

```
Private Sub AddDriverSession()

Dim cs As New IviConfigStore
Dim ha As IviHardwareAsset
Dim hadup As IviHardwareAsset
Dim ds As IviDriverSession
Dim vn As IviVirtualName
Dim vr As IviVirtualRange
Dim dc As IviDataComponent
Dim dcb As IviBoolean
Dim ln As IviLogicalName

'// Deserialize the master configuration store.

'On Error GoTo DeserializeError
cs.Deserialize (cs.MasterLocation)
On Error GoTo 0

'// Create the Hardware Asset and add it to the global hardware asset
'//  collection
Set ha = New IviHardwareAsset
ha.Name = "Scope 5"
ha.Description = "GT4010 Scope, test station 5"
ha.IOResourceDescriptor = "GPIB0::12::INSTR"
On Error GoTo DuplicateHardwareAsset
cs.HardwareAssets.Add ha
On Error GoTo 0

'// Create the Session fill in the Session object properties
Set ds = New IviDriverSession
ds.Name = "Scope5"
ds.Description = "Driver session forscope at test station 5"
ds.Cache = False
ds.DriverSetup = ""
ds.InterchangeCheck = True
ds.QueryInstrStatus = False
ds.RangeCheck = False
ds.RecordCoercions = False
ds.Simulate = True

'// Add the Hardware Asset reference to the Session
Set ds.HardwareAsset = cs.HardwareAssets.Item("Scope 5")

'// Create the Virtual names for the Session.  The creates the following
'//  mappings: Analog -> C1, 1 -> C2, 2 -> C3, and 3 -> C4.
Set vn = New IviVirtualName
vn.Name = "Analog"
vn.MapTo = "C1"
ds.VirtualNames.Add vn
Set vn = New IviVirtualName
vn.MapTo = "C"
vn.Name = ""
Set vr = New IviVirtualRange
vr.Name = "Virt CH 1-3"
vr.Max = 3
vr.Min = 1
vr.StartingPhysicalIndex = 2
vn.VirtualRanges.Add vr
ds.VirtualNames.Add vn
```

```
'// Add the Software Module reference to the Session.  The configuration
'//  server will automatically copy all data components with UsedInSession
'//  = "Required" or "Optional" to the session's data components, and
'//  change the ReadOnly property to False.
Set ds.SoftwareModule = cs.SoftwareModules.Item("gt40xx")

'// Change the default values for Data Components for the Session, if needed
Set dcb = ds.DataComponents.Item("Trace")
dcb.Value = True

cs.DriverSessions.Add ds

'// Create the Logical Name and add it to the global logical name collection
Set ln = New IviLogicalName
ln.Name = "Bob"
ln.Description = "Logical name for Scope at test station 5"
Set ln.Session = ds
On Error GoTo DuplicateLogicalNames
cs.LogicalNames.Add ln
On Error GoTo 0

Exit Sub

'// Error handler for duplicate hardware assets

DuplicateHardwareAsset:
Set hadup = cs.HardwareAssets.Item("Scope 5")
If ha.IOResourceDescriptor = hadup.IOResourceDescriptor _
  Then
   Resume Next  '// The hardware asset already exists - we just move forward
  End If
'// The hardware asset "Scope 5" refers to a different IO Resource.
'//  Handle this error appropriately.
Exit Sub

'// Error handler for duplicate Logical Names

DuplicateLogicalNames:
'// The logical name "Bob" already exists.
'//  Handle this error appropriately.
Exit Sub
End Sub
```

The XML file created by this example, with extra line breaks, looks like:

```
<IviConfigStore xmlns:dt="urn:schemas-microsoft-com:datatypes">
<Name>IVI Configuration Server</Name>
<Description>The IVI Configuration Server allows access to and modification of
an IVI configuration store</Description>
<Vendor>IVI Foundation, Inc</Vendor>
<Revision>1.3.0.3</Revision>
<SpecificationMajorVersion>1</SpecificationMajorVersion>
<SpecificationMinorVersion>0</SpecificationMinorVersion>
<MasterLocation>C:\Program
Files\IVI\Data\IviConfigurationStore.xml</MasterLocation>
<ProcessDefaultLocation>
</ProcessDefaultLocation>
<ActualLocation>
</ActualLocation>
<PublishedAPIs>
<IviPublishedAPI id="p1">
```

```
<Name>IviDriver</Name>
<MajorVersion>2</MajorVersion>
<MinorVersion>0</MinorVersion>
<Type>IVI-COM</Type>
</IviPublishedAPI>
<IviPublishedAPI id="p2">
<Name>IviScope</Name>
<MajorVersion>2</MajorVersion>
<MinorVersion>0</MinorVersion>
<Type>IVI-COM</Type>
</IviPublishedAPI>
</PublishedAPIs>
<SoftwareModules>
<IviSoftwareModule id="p3">
<Name>gt40xx</Name>
<Description>IVI-COM Specific Instrument Driver for GT40xx family of
oscilloscopes</Description>
<DataComponents>
<IviBoolean id="p4">
<Name>Trace</Name>
<Description>If True, tracing is on, if False, tracing is off</Description>
<ReadOnly>1</ReadOnly>
<UsedInSession>Required</UsedInSession>
<Type>Boolean</Type>
<HelpContextID>0</HelpContextID>
<HelpFilePath>
</HelpFilePath>
<SoftwareModuleKey>
</SoftwareModuleKey>
<Value>0</Value>
</IviBoolean>
</DataComponents>
<ModulePath>
</ModulePath>
<Prefix>gt40xx</Prefix>
<ProgID>gt40xx.gt40xx</ProgID>
<SupportedInstrumentModels>gt4000,gt4001,gt4010,gt4011,gt4012</SupportedInstrum
entModels>
<PhysicalNames>
<IviPhysicalName id="p5">
<Name>C</Name>
<RCName>Channel</RCName>
<PhysicalNames/>
<PhysicalRanges>
<IviPhysicalRange id="p6">
<Name>C Range 1</Name>
<Max>4</Max>
<Min>1</Min>
</IviPhysicalRange>
</PhysicalRanges>
</IviPhysicalName>
</PhysicalNames>
<PublishedAPIs>
<IviPublishedAPI idref="p1"/>
<IviPublishedAPI idref="p2"/>
</PublishedAPIs>
</IviSoftwareModule>
</SoftwareModules>
<HardwareAssets>
<IviHardwareAsset id="p7">
<Name>Scope 5</Name>
<Description>GT4010 Scope, test station 5</Description>
```

```
<DataComponents/>
<IOResourceDescriptor>GPIB0::12::INSTR</IOResourceDescriptor>
</IviHardwareAsset>
</HardwareAssets>
<DriverSessions>
<IviDriverSession id="p8">
<Name>Scope5</Name>
<Description>Driver session forscope at test station 5</Description>
<DataComponents>
<IviBoolean id="p9">
<Name>Trace</Name>
<Description>If True, tracing is on, if False, tracing is off</Description>
<ReadOnly>0</ReadOnly>
<UsedInSession>Required</UsedInSession>
<Type>Boolean</Type>
<HelpContextID>0</HelpContextID>
<HelpFilePath>
</HelpFilePath>
<SoftwareModuleKey>
</SoftwareModuleKey>
<Value>1</Value>
</IviBoolean>
</DataComponents>
<IviHardwareAsset idref="p7"/>
<IviSoftwareModuleRef idref="p3"/>
<VirtualNames>
<IviVirtualName id="p10">
<Name>
</Name>
<MapTo>C</MapTo>
<VirtualRanges>
<IviVirtualRange id="p11">
<Name>Virt CH 1-3</Name>
<Max>3</Max>
<Min>1</Min>
<StartingPhysicalIndex>2</StartingPhysicalIndex>
</IviVirtualRange>
</VirtualRanges>
</IviVirtualName>
<IviVirtualName id="p12">
<Name>Analog</Name>
<MapTo>C1</MapTo>
<VirtualRanges/>
</IviVirtualName>
</VirtualNames>
<SoftwareModuleName>gt40xx</SoftwareModuleName>
<Cache>0</Cache>
<DriverSetup>
</DriverSetup>
<InterchangeCheck>1</InterchangeCheck>
<QueryInstrStatus>0</QueryInstrStatus>
<RangeCheck>0</RangeCheck>
<RecordCoercions>0</RecordCoercions>
<Simulate>1</Simulate>
</IviDriverSession>
</DriverSessions>
<Sessions>
<IviDriverSession idref="p8"/>
</Sessions>
<LogicalNames>
<IviLogicalName id="p13">
<Name>Bob</Name>
```

```
<Description>Logical name for Scope at test station 5</Description>
<IviDriverSession idref="p8"/>
</IviLogicalName>
</LogicalNames>
</IviConfigStore>
```

Appendix B: .NET Configuration Server API and IVI-COM Driver Example

This example is written using the native .NET Configuration Server.

The subject of this example is an imaginary IVI-COM instrument specific driver. The driver supports a family of oscilloscopes from GizmoTronics , Inc. The driver supports the IVI inherent interfaces and the IVI scope class-compliant interfaces. In addition to the standard IVI configuration properties, the driver can be configured from the configuration server to turn tracing on and off. The driver supports four channels that it knows as “C1” through “C4”.

All examples are in C#. They may be abbreviated to emphasize configuration server use.

B.1 Assembly References

After creating a source file for the example code, add the native IVI.NET Configuration Server to the example project, and add the following references to the code:

```
using System;
using System.IO;
using System.Text;
using Ivi.ConfigServer;
```

B.2 Creating an Empty Configuration Store for the Example

Do not use the configuration store file that is installed with the IVI Shared Components. Instead, create a file just for use with the example: This section shows a method that creates a new file in the temp folder.

```
private static string exampleStore { get; } = Path.Combine(
                                                    @”C:\Temp”,
                                                    "ExampleStore.xml");

private void CreateEmptyConfigStore()
{
   if (File.Exists(exampleStore))
      File.Delete(exampleStore);

   var server = new ConfigStore();
   server.Save(exampleStore);
}
```

B.3 Adding a Software Module

This section shows the configuration server code to install a software module (in this case, an IVI-COM specific instrument driver).

```
private void AddSoftwareModule()
{
    const string moduleName = "gt40xx";

    ConfigStore server;
```

```
PublishedApi apiDriver;
PublishedApi apiScope;

try
{
    server = ConfigStore.Load(exampleStore);
}
catch (ArgumentException)
{
    // Handle the exception appropriately.
    throw;
}

if (server.SoftwareModules.ContainsKey(moduleName))
    server.SoftwareModules.Remove(moduleName);

if (server.PublishedApis.ContainsKey(
        IviPublishedApiName.IviDriver,
        IviPublishedApiType.IviCom,
        2, 0))
{
    apiDriver = server.PublishedApis[
                    IviPublishedApiName.IviDriver,
                    IviPublishedApiType.IviCom,
                    2, 0];
}
else
{
    apiDriver = new PublishedApi(
                        IviPublishedApiName.IviDriver,
                        IviPublishedApiType.IviCom,
                        2, 0);
    server.PublishedApis.Add(apiDriver);
}

if (server.PublishedApis.ContainsKey(
        IviPublishedApiName.IviScope,
        IviPublishedApiType.IviCom,
        2, 0))
{
    apiScope = server.PublishedApis[
                   IviPublishedApiName.IviScope,
                   IviPublishedApiType.IviCom,
                   2, 0];
}
else
{
    apiScope = new PublishedApi(
                       IviPublishedApiName.IviScope,
                       IviPublishedApiType.IviCom,
                       2, 0);
    server.PublishedApis.Add(apiScope);
}

server.SoftwareModules.Add(
    new IviComSoftwareModule(
        moduleName,
```

```
            moduleName,
            moduleName + "." + moduleName,
            apiDriver,
            apiScope,
            new PhysicalName("C",
                new PhysicalRange("C Range 1", 1, 4))
            { RCName = "Channel" },
            new IviBoolean("Trace", false, true, SessionUsage.Required)
            { Description = "If True, tracing is on, otherwise off" })
        {
            Description = "IVI-COM Specific Instrument Driver for"
                        + " GT40xx family of oscilloscopes",
            SupportedInstrumentModels = "gt4000,gt4001,gt4010,gt4011,gt4012"
        });

    server.Save(exampleStore);
}
```

B.4 Adding a Driver Session

The section shows the configuration server code to add a driver session that uses the previously added software module. A logical name (“Scope5”) will refer to the session. The session will refer to a hardware asset whose resource descriptor is “GPIB0::12::INSTR”. It will also provide logical names for the software modules physical names and configure the values of the trace data component.

Bear in mind that most end-users will use a GUI to edit the configuration store, but some users may choose to write code like this – for example, as part of a test system. In any case, this example code is meant to illustrate the kinds of configuration server entries that must be made. It is not meant to apply to every Driver Session definition.

```
private void AddDriverSession()
{
    ConfigStore server;

    try
    {
       server = ConfigStore.Load(exampleStore);
    }
    catch (ArgumentException)
    {
        // Handle the exception appropriately.
        throw;
    }

    if (!server.HardwareAssets.ContainsKey("Scope 5"))
    {
        server.HardwareAssets.Add(
            new HardwareAsset("Scope 5", "GPIB0::12::INSTR")
            { Description = "GT4010 Scope, test station 5" });
    }
    else if (server.HardwareAssets["Scope 5"].IOResourceDescriptor !=
       "GPIB0::12::INSTR")
    {
        // Handle this condition appropriately.
    }
```

```
    if (server.DriverSessions.ContainsKey("Scope 5"))
        server.DriverSessions.Remove("Scope 5");

    server.DriverSessions.Add(
        new DriverSession(
            "Scope5",
            server.HardwareAssets["Scope 5"],
            new VirtualName("Analog", "C1"),
            new VirtualName("Digital", "C",
                new VirtualRange("Virt CH 1-3", 1, 3, 2)),
            server.SoftwareModules["gt40xx"])
        {
            Description = "Driver session for scope at test station 5",
            DriverSetup = "",
            Cache = false,
            InterchangeCheck = true,
            QueryInstrStatus = false,
            RangeCheck = false,
            RecordCoercions = false,
            Simulate = true
        } );

    var trace = (IviBoolean)
                server.DriverSessions["Scope5"].DataComponents["Trace"];
    trace.Value = true;

    if (!server.LogicalNames.ContainsKey("Bob"))
    {
        server.LogicalNames.Add(
            new LogicalName("Bob",
                server.DriverSessions["Scope5"]));
    }
    else
    {
        // Handle this condition appropriately.
    }

    server.Save(exampleStore);
}
```