Phase One CaptureCore SDK 6.2

Interface Revision 2

2011-07-08 version 622.2.50741

PHASEONE

PHASEONE

1 Introduction

Welcome to the Phase One CaptureCore SDK. CaptureCore is a software interface for communication with all Phase One digital backs and cameras. It is specifically for use in the capturing and transferring of images to a computer connected to a digital back or camera, as well as the setup and synchronization of device settings. It consists of a set of generalized object-oriented classes that support the .NET and ObjC development environments on Microsoft Windows and Apple Mac OS operating systems.

1.1 Design Concepts

CaptureCore is designed to be general, cross-platform, object-oriented, and thread-safe.

The primary design focus for CaptureCore was generalization. CaptureCore provides a common interface for different models of digital backs and cameras. For the application developer, this device-independence means that the same code will work with all supported devices. In addition, CaptureCore has been designed to support a broad range of applications. To provide this generality without losing functionality, CaptureCore provides methods by which an application can check if a specific device supports a feature it wishes to use.

CaptureCore is cross-platform, supporting the development of applications on both Microsoft Windows and Apple Mac OS operating systems. It currently supports all .NET development environments on Microsoft Windows (C#, Managed C++, Visual Basic, and others), and supports ObjC development on Apple Mac OS. CaptureCore adheres as much as possible to the different coding styles for each development environment, while still maintaining a common interface across platforms.

CaptureCore is object-oriented, making it easy to use in today's object-oriented development environments. In addition, reference-counting is used to automate object destruction.

All classes, methods, and properties in CaptureCore are thread-safe. Thus the application developer is free to use multi-threading without additional code.

It has been challenging and fun to design CaptureCore and we hope that you enjoy using it in your application.

1.2 Supported Environments

CaptureCore supports all 32-bit and 64-bit versions of Windows XP, Windows Vista and Windows 7. It also supports Apple Mac OS 10.5 and later.

1.3 Supported Devices

The following devices are supported.

Manufacturer	Model
Phase One	IQ180, IQ160, IQ140 P65+, P40+ P45+, P45+ BW, P45, P30+, P30, P25+, P25, P21+, P21, P20+, P20 H 101, H 25, H 20, H 10-11M, H 10-6M, H 5 Achromatic, LightPhase

1.4 SDK Contents

The files included in the SDK are organized in the following folder hierarchy.

┌/Doc	Documentation files for all platforms.
├/Win	Microsoft Windows specific files.
│ ├/Bin │ │ │ │ │ └/Drivers	• Redistributable binary files that provide CaptureCore functionality. • Driver files. Some devices require a driver to be installed.
│ └/Samples	• Windows sample applications.
│ └/Net	• .Net sample applications
│ └/C#	• C# sample applications
│ └/SimpleCapture	• C# simple command-line capture application.
└/Mac	Apple Mac OS specific files.
├/Bin │	• Redistributable binary files that provide CaptureCore functionality.
└/Samples	• Mac OS sample applications.
└/ObjC	• ObjC sample applications
└/SimpleCapture	• ObjC simple command-line capture application.

1.5 Sample Applications

1.5.1 SimpleCapture

The *SimpleCapture* sample application is a simple command-line capture application. It demonstrates basic capture functionality, such as enumerating capture devices, plug & play, capturing images, requesting a remote capture, editing a camera property, and handling of errors, events and progress status.

2 Overview

CaptureCore consists of a set of class interfaces representing cameras, images, manufacturers, capabilities, properties, and so on. These are organized in a class hierarchy and are derived from common base classes – thus many classes share a common set of methods.

Objects (instances of these class interfaces) are organized in an object hierarchy, where most objects are owned by a parent object, and can have one or more child objects. Reference-counting is used to automate object destruction, though in both .NET and ObjC this is handled for the developer by the development runtime. There is a single instance of a top-level object (*ICaptureCore*) at the top of the hierarchy, which is retrieved by a call to the global method *GetCaptureCore*.

All method calls (including .Net properties) can throw exceptions in the event of an error condition, such as an invalid parameter, or a communication error with a capture device. In addition, background threads which encounter errors can inform the application of the error via events.

Some objects have attributes called capabilities and properties. A capability describes a conditional feature that is sometimes available, allowing an application to test if an object has a specific capability before attempting to use it. Properties are settings and information for the object. Some properties are read-only, and describe things such as a capture device's serial number or model name. Other properties are writeable, such as exposure ISO, and may have a setting descriptor object that describes which values or the range that they can be set to. Properties are designed to be easily presented to the user via common user-interface controls, such as an edit, drop-down list, or combination control.

Objects can send events to an application defined event receiver, informing of errors, captured images, removed devices, and so on. Progress status is also given for tasks that can take some time.

2.1 Namespaces

In development environments that support namespaces (.NET), CaptureCore classes and enumerations are declared in P1.CaptureCore (P1::CaptureCore).

2.2 Object Hierarchy

Objects in CaptureCore are organized in a object/data hierarchy embodying the ownership relationship between objects. Most objects are a child of a parent object that owns the child object. The top-level object of the hierarchy is a single instance of *ICaptureCore*, which is returned by calling the global method *GetCaptureCore*.

This single *ICaptureCore* object is the parent of *ICaptureProvider* objects, which represent a capture device manufacturer, such as Phase One, or a specific family of devices from a manufacturer. *ICaptureProvider* objects are parents to *ICamera* objects, which represent a physically connected camera or capture device. *ICamera* objects are parents to *ICaptureImage* objects, which contain a captured image. The entire object hierarchy is shown below in Figure 1.

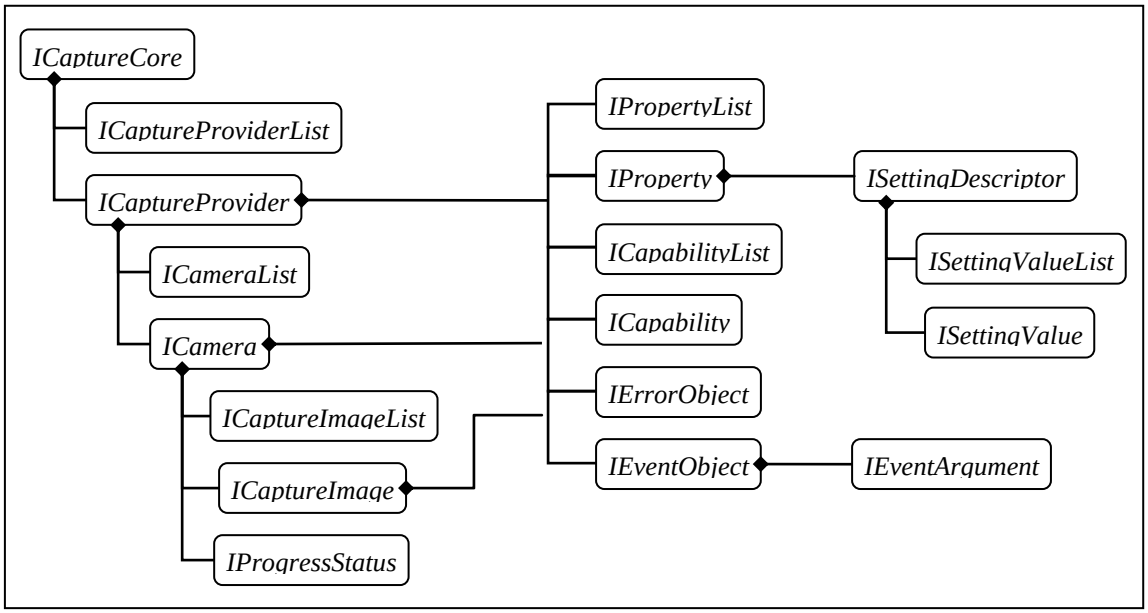

Figure 1: CaptureCore Object Hierarchy

There are three principle CaptureCore objects that an application will interact with: *ICaptureProvider*, *ICamera*, and *ICaptureImage*, which represent manufacturers, devices, and images respectively. These three objects share common traits, which they inherit from *ICaptureObject*. All three have capabilities and properties, can broadcast events, and support background error reporting.

ICaptureProvider objects provide the application with a list of attached capture devices (*ICamera* objects), and send events when new devices are connected or existing devices are disconnected. There is an *ICaptureProvider* object for every device API supported by CaptureCore.

The *ICamera* class is the fundamental class of CaptureCore. *ICamera* objects allow the application to interact with attached capture devices, and to control the capture and transfer of images. They receive and store captured images (*ICaptureImage* objects), sending an event whenever a new image is available. Many device settings can also be set via an *ICamera* object. There is an *ICamera* object for every connected capture device supported by CaptureCore.

ICaptureImage objects encapsulate the images captured by devices and transferred to the host. They provide access to image properties and image data, which can be saved to a file or copied to an application buffer. A thumbnail (*ICaptureImageThumbnail*) can also be retrieved for some image types.

2.3 Class Hierarchy

CaptureCore objects are instances of CaptureCore class interfaces. It is through these class interfaces that CaptureCore is accessed by an application. These class interfaces are organized in a class hierarchy, where most classes are derived from common base classes. In this way classes share common methods and functionality. The class hierarchy is shown below in Figure 2.

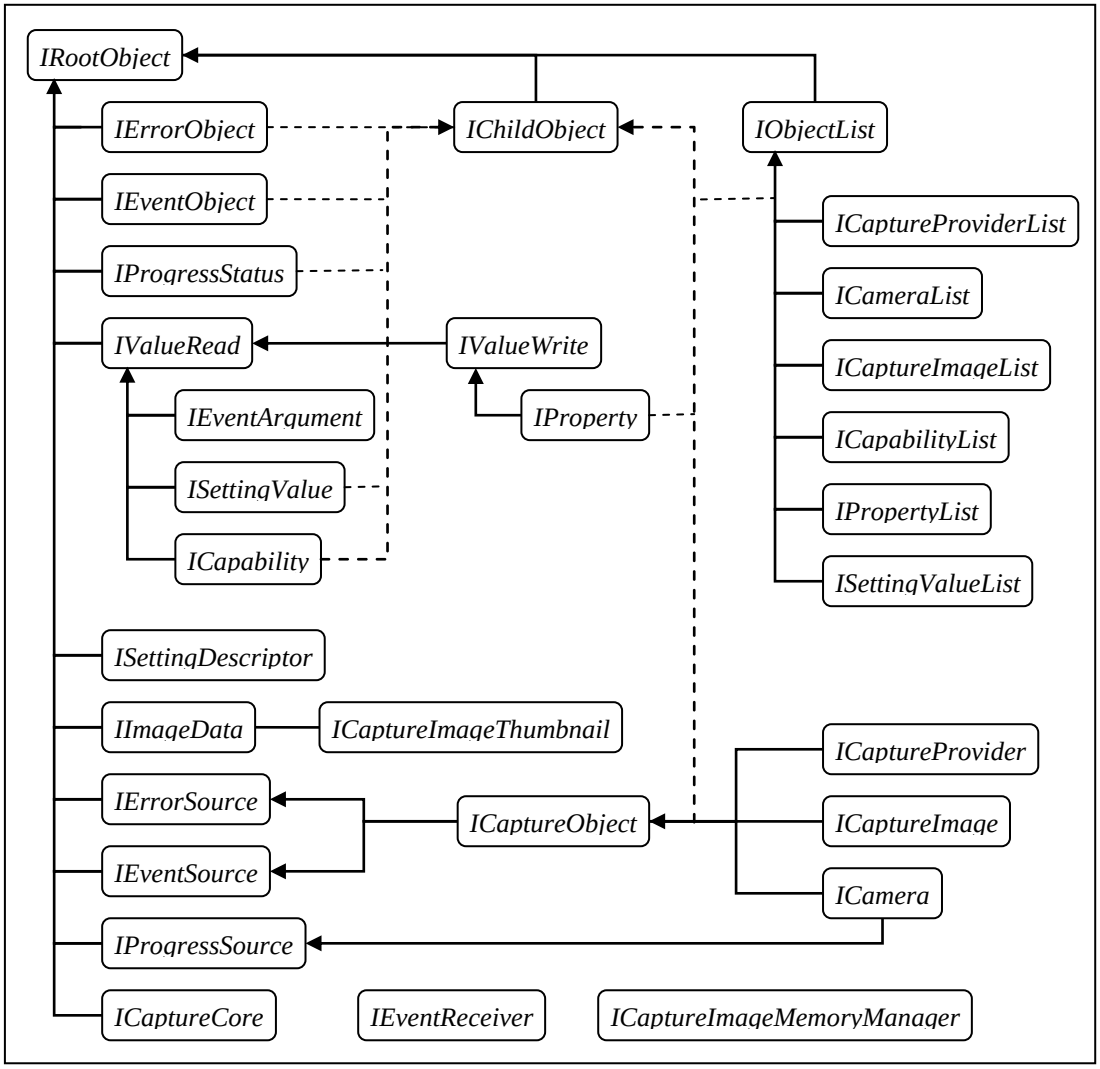

Figure 2: CaptureCore Class Hierarchy

2.4 Reference Counting

Reference counting is used throughout CaptureCore to automate the life-cycle of all objects. As long as a reference exists to an object, it exists and its resources are not released by CaptureCore. When the last reference to an object is released, the object is automatically destroyed. Reference counting is automated in .Net and ObjC, so applications developed in those environments do not need to explicitly retain or release references.

A parent object also keeps a reference to all its child objects. When a parent no longer needs a child object, it releases its reference to the child, and that child no longer has a parent. Normally the reference held by the parent is the last reference to the child object and it is immediately destroyed. However, if the application also has a reference to the child object, the child object continues to exist and becomes an orphan. Calls to the child objects *Parent* method will return a NULL reference. Once the application releases its last reference to an orphaned child object, it is finally destroyed.

2.5 Multithread Applications

All CaptureCore objects and their methods and properties are thread-safe. Thus applications don't generally need to use any thread synchronization to use CaptureCore. However, when several calls to a CaptureCore object are desired to be atomic, the application can still find it

useful to use a locking mechanism around some CaptureCore calls. It is up to the application to provide this locking mechanism.

For example, if an application wishes to check if an object is open before calling a method that only succeeds if it is open, it may wish to put a lock around these two calls, to ensure that no other application thread closes the object between these two calls. This also requires placing a lock around all calls to close the object in the application.

CaptureCore uses multiple threads to perform several background tasks. Calls to application provided event receivers, and other callback interfaces, are performed by background threads. Thus application implementations of callback interfaces need to be thread-safe with regards to other code in the application.

2.6 Errors, Exceptions and Return Values

CaptureCore objects generally communicate errors by throwing exceptions. Any method in CaptureCore can throw an exception if an error occurs. An application should be prepared to handle exceptions from any call to a CaptureCore method. CaptureCore exception objects are derived from the *IErrorObject* class, which contains methods for determining which error occurred, and description and detail strings for the error. In addition, for each development environment, CaptureCore exception objects are also derived from the native exception class for that environment, such as *System.Exception* in .Net and *NSException* in ObjC.

Objects of classes derived from *IErrorSource*, can also report an error though the *GetError* method of *IErrorSource*. This allows such objects to report errors that occur on background threads, that is errors that do not arise directly from a method call. If an *IErrorSource* object is also derived from *IEventSource*, a *kEventId_Error* event is posted whenever there is a new error that can be retrieved by calling *GetError*. Applications should monitor objects derived from *IErrorSource* for the *kEventId_Error* event, and retrieve the error by calling *GetError*.

CaptureCore does not use return values as a mechanism for reporting errors. However, many methods in CaptureCore return pointers or references to objects, and these methods will return a NULL reference, when the requested object is not available. This is not an error, but a normal return value for these methods. In the event of a true error, these methods will still throw an exception. Applications should expect to get a NULL value for any object pointer or reference returned from a CaptureCore method, and test for NULL before using the object.

2.7 Value Classes and Types

Several CaptureCore classes represent simple values, such as numbers or strings. These value classes are derived from either *IValueRead* or *IValueWrite*, such as the classes *ICapability* and *IProperty*, which represent an object's capabilities and properties. *IValueRead* classes represent read-only values, whereas *IValueWrite* classes represent values that can both be read and written. *IValueWrite* inherits from *IValueRead*.

Values represented by *IValueRead* or *IValueWrite* can be one of several different value types, defined by the enumeration *EnumValueType*: a Boolean, 32- and 64-bit signed and unsigned integers, a floating point, a string, or an enumeration. An enumeration value type is both a 32-bit signed/unsigned integer and a string at the same time. Regardless of the actual value type of a value, a string representation can be retrieved for all values. The value type of a value class can be retrieved by the *ValueType* method of *IValueRead* (and *IValueWrite*).

An application reads or writes the value represented by a value class by calling one of the *GetValue* and *SetValue* methods defined by *IValueRead* and *IValueWrite*. These methods will throw an exception if an incompatible value type is passed to the method.

For many classes, and especially for capabilities and properties, the value type of a value object is not predefined. Applications should not expect a value object to be of any particular value type, and should attempt to handle all value types for each and every value object.

2.8 Capabilities

Objects of some classes, specifically those derived from *ICaptureObject*, can have capabilities. A capability describes a conditional feature that is sometimes available. Capabilities allow an application to test if an object has a feature before attempting to use it. Capabilities are different from properties in that capabilities are always read-only and are usually only used by the application for conditionally enabling functionality.

Capabilities are represented by the *ICapability* class, which is derived from *IValueRead*, and are organized in a list (*ICapabilityList*) for each class that supports capabilities. They are uniquely identified by a capability ID (enumeration value), and a specific capability can be retrieved by iterating the capability list or calling the *GetCapability* method.

Capabilities can be of any value type (*EnumValueType*), such as a number or a string, but generally they will be a Boolean value. There is no guarantee that a specific capability will be of a specific value type. An application should be prepared to handle different value types for every capability. It is recommended that an application always tests a capability's value type before using it. When a capability changes value, the owning class will often post an event to indicate this.

Which capabilities are present is determined by the implementation of the class which has capability support. The application should not use a feature if the capability representing the feature is not present for a specific class, or if the capability indicates that the feature is not supported. Doing so may result in an exception, though often the functionality will just do nothing.

See the Capability Reference section for a list over all capabilities supported by different classes.

2.9 Properties

Objects of some classes, specifically those derived from *ICaptureObject*, can have properties. A property describes a setting or some user information for the object. Some properties are read-only, and describe things such as a capture device's serial number or model name. Other properties are writeable, such as exposure ISO, and may have a setting descriptor object that describes which values or the range that they can be set to. Properties are designed to be easily presented to the user via common user-interface controls, such as an edit, drop-down list, or combination control.

Properties are represented by the *IProperty* class, which is derived from *IValueWrite*, and are organized in a list (*IPropertyList*) for each class that supports properties. They are uniquely identified by a property ID (enumeration value), and a specific property can be retrieved by iterating the property list or calling the *GetProperty* method.

Properties can be of any value type, such as a number or a string. There is no guarantee that a specific property will be of a specific value type. An application should be prepared to handle different value types for every property. It is recommended that an application always tests a property's value type before using it. When a property changes value, the owning class will often post an event to indicate this.

Properties may have a setting descriptor object (*ISettingDescriptor*) that describes the values or the range that they can be set to. Even if the property is read-only it may have a setting descriptor. When a property is set to a value by an application, the value is validated with the settings described by its setting descriptor, and an exception is thrown if the value is not allowed. If no setting descriptor object is present, then there is no limit to what the property can be set to (within the limits of the value type of the property). Simple setting descriptors provide a minimum/maximum range for the property. Others provide a list of values that the property can be set to. This list can be select only, where only values in the list are allowed, or it could just represent commonly used values, allowing the property to be set to values not in the list. When a setting descriptor changes value, the owning class will often post an event to indicate this.

Some properties have default values (defined by the setting descriptor) and can be restored to their default. Properties can have an undefined value (see *IsUndefined*), if the current value is unknown or not available. This often just indicates that the property has not been set yet. The property can also be disabled (see *IsDisabled*), if the property is currently not accessible or unavailable for some reason. A disabled property is automatically read-only, and may even throw an exception if read. A property can be disabled or enabled at any point in time, depending upon the cause. An event is generally posted when this occurs.

Generally, properties are automatically synchronized with the source or device that owns the property, but not always, especially if it will negatively affect performance, or if it is not technically possible to do so. In this case, an application can call the *Refresh* method to request a manual synchronization from the property source.

When using properties, an application has two possibilities: it can request specific properties using their property IDs and handling if the property doesn't exist, or it can support all properties exposed by an object, without even examining the property ID. The first method allows specific properties to be picked out and presented to the user. The application must still be prepared to handle all value types for each property. The second method displays all available properties in a generic manner, using the strings contained within the property object to describe the property. Since an application may not be aware of all possible property IDs at the time of creation, the second method is more generic and future compatible. Both methods can of course be combined, handling some properties differently, while still listing all remaining.

See the Property Reference section for a list over all properties supported by different classes.

2.10 Events

Objects derived from the *IEventSource* class can post events. Events typically represent changes in state for the object and are represented by an *IEventObject* object. Each event is uniquely identified by an event ID (enumeration value). The possible event IDs for each object are described in each class description. Events can also include event arguments, which are simple values represented by an *IEventArgument* object.

An application subscribes to events by adding an event receiver to an *IEventSource* object. An event receiver is an application implemented class that implements the *IEventReceiver* interface. The *IEventReceiver* interface has one method: *OnEvent*, which is called when delivering events to the event receiver. The application is free to implement the *IEventReceiver* interface in combination with other interfaces, or as part of another application class. Typically, an application implements the *IEventReceiver* interface for each class that will receive events.

Applications call the *AddReceiver* or *RemoveReceiver* methods of an *IEventSource* object to subscribe or unsubscribe to events from that object. An application can chose to subscribe to specific events, by passing the desired event ID to *AddReceiver*, or to subscribe to all events by passing *kEventId_All*.

All events are asynchronous. That is events are delivered by a separate thread than that which posted the event. Thus an event receiver will receive the event shortly after it occurred. This delay is usually very small, in the order of microseconds, but can vary depending upon how busy the computer is. An event dispatch thread is created for each subscribed event receiver. Thus an event receiver cannot delay the delivery of events to other event receivers. Events for a single event receiver are always delivered sequentially in order, and an event receiver will not receive a new event before returning from a previous call to *OnEvent*.

In order to keep the flow of events as timely as possible, it is the responsibility of the application's *IEventReceiver* implementation to handle each event promptly, and to not call any methods that may block indefinitely. Further, a call to *RemoveReceiver* will block until an event receiver's *OnEvent* method has returned, and this could lead to a deadlock situation. Thus, applications should avoid waiting on threads in their *OnEvent* implementation, if the same thread may call *RemoveReceiver*. For example, if an *OnEvent* implementation waits on the main thread to perform some action, and the main thread calls *RemoveReceiver*, then both threads end up waiting on each other, and a deadlock occurs.

Although events are dispatched sequentially to each receiver, there is no guarantee regarding the order that events are sent from any part of CaptureCore, due to the multithreaded nature of CaptureCore. Thus the application should avoid making any assumptions about the order of events, or have cross-dependencies between events. For example, a progress event of 100% may not always be sent, before an image arrived event, and may come slightly out of order or not at all. Further, an image arrived event may not be sent, following progress events for that image, if the image is cancelled before completion.

Generally, each event is self-contained, and is designed to communicate only a single piece of information, and event handling code should be designed similarly. For example, only use progress events to update a progress control, without any additional actions. Since event delivery can be delayed, the state reported by the event may not be current. Event handling code may wish to verify the state reported by an event, before taking the appropriate action.

2.11 Progress

Objects derived from *IProgressSource*, such as *ICamera* objects, can inform the application of the progress status of different tasks. The progress status of a task is described by *IProgressStatus* objects, which are queued by the *IProgressSource* object. An *IProgressSource* object posts an *kEventId_ProgressUpdate* event when a new *IProgressStatus* object is queued, which an application can retrieve by calling the *GetProgress* method of the *IProgressSource* class.

More than one task can be active at the same time. The *Id* and *Instance* members of *IProgressStatus* can be used to differentiate between different progress tasks. *Id* returns an enumeration value which specifies the kind of progress the *IProgressStatus* object describes, such as image capture or file saving progress. *Instance* returns an unique number for each progress task. No two progress tasks will have the same instance number.

IProgressStatus objects contain many members which provide string descriptions of the progress task, as well as how much of the task is completed, and how long the task has been

running. In addition, it is possible for some tasks to cancel the task, by calling the Cancel member of the *IProgressStatus* class.

IProgressStatus objects are queued, and therefore there is a time lag between when the status was generated and when it is retrieved by the application. The application should therefore try to handle progress status events as quickly as possible, to minimize this lag.

The application should be careful to avoid making any assumptions about the delivery of progress status for a specific task. For example a task may not always reach 100%, due to an error or if it is cancelled. Further, no guarantee is made about the order of progress events with other CaptureCore events. Progress status should be regarded as informational only, and used for display purposes and not to control the state of the application.

2.12 Log File and Cache Folder

CaptureCore can be setup to log messages regarding the internal activities of CaptureCore to a log file. Log messages can be useful for tracking the internal operations carried out by CaptureCore, as well as logging errors and warnings. Log messages include the date, time and thread ID for each message. The *LogMsgFileName* methods of the *ICaptureCore* class can be used to set or clear the log file's filename. If not filename is set, which is the default state, then no log file is created. The application is responsible for ensuring that the path to the log file is valid, and that the application has file creation rights for the provided path. Note that CaptureCore appends to an existing file, so given enough time the log file can become quite large.

CaptureCore can cache certain object data and settings in order to increase performance and provide setting persistency. Cache files are created for each device within an application specified cache folder. The *CacheFolderName* methods of the *ICaptureCore* class can be used to set or clear the cache folder path. If no cache folder is specified, which is the default, then no cache files are used. It is highly recommended to define a cache folder for applications using CaptureCore. The application is responsible for ensuring that the provided folder path is valid, and that the application has both folder an file creation rights.

2.13 Generality and Future Compatibility

CaptureCore supports many different capture devices from several vendors. In order to handle all the possible variations that exist now or in the future, CaptureCore is designed with a focus on generality. Many features of CaptureCore are dynamic, such as capabilities, properties, and value types.

An application will generally need to do a bit more work in order to support this generality, such as checking for the value type of a value object, or testing for a capability before using a specific feature. However, the advantage is great. Once the application code is written, few if any changes will be necessary to support new devices in the future, or to be compatible with future changes in CaptureCore.

It is highly recommended that application developers embrace the generality of CaptureCore when implementing the applications that will use it. They will gain much in terms of future compatibility. Assumptions about how CaptureCore works, based upon observation, should be avoided. For example, one cannot assume that the value type of a specific property will always be the same for all devices or all versions of CaptureCore. Nor can one assume that events will always arrive or that they will arrive in a certain order.

It is also a good practice to always test the return values of every call, and to be prepared for exceptions from any call.

2.14 Development Environment Differences

There are a few differences from the general documentation for each of the development environments.

2.14.1 .Net

All exceptions thrown by CaptureCore are instances of the *CaptureCoreException* class. A *CaptureCoreException* object implements the *IErrorObject* interface and also inherits from *System.Exception*. This is because .Net requires all exception objects to be derived from *System.Exception*. Thus *CaptureCoreException* objects provide both the *IErrorObject* methods documented in this document and the .Net *System.Exception* class methods.

2.14.2 ObjC

All CaptureCore class names are preceded by a *P1CaptureCore_* prefix to avoid conflicting with other names in the global namespace of the application. This is done since ObjC does not support namespaces.

3 Reference

3.1 GetCaptureCore

GetCaptureCore initializes CaptureCore and returns the top-level *ICaptureCore* object in the CaptureCore object hierarchy, through which all CaptureCore functionality is accessed.

Syntax

.Net

C#	`static ICaptureCore  CaptureCoreEntry.GetCaptureCore()`
C++	`static ICaptureCore^ CaptureCoreEntry::GetCaptureCore()`
VB	`Shared Function CaptureCoreEntry.GetCaptureCore As ICaptureCore`

ObjC

```
@interface P1CaptureCore_CaptureCore
+ (id) getCaptureCore
```

Return Value

The top-level *ICaptureCore* object of the CaptureCore object hierarchy. A NULL reference is returned if CaptureCore cannot be initialized. There is only a single *ICaptureCore* object in the CaptureCore object hierarchy, so subsequent calls to this function will return the same object.

3.2 ICaptureCore (P1CaptureCore_CaptureCore)

The *ICaptureCore* class provides access to all the functionality of CaptureCore. Only a single instance of an *ICaptureCore* object exists, and is retrieved by calling *GetCaptureCore*.

ICaptureCore is a parent to *ICaptureProvider* (*ICaptureProviderList*) objects.

Members

`Version`	Returns the version string of the CaptureCore assembly/framework file.
`Revision`	Returns the interface revision number of the CaptureCore assembly/framework file.
`Terminate`	Releases all CaptureCore resources. Once *Terminate* is called, CaptureCore can no longer be used.
`GetCaptureProviderList`	Returns a list of all supported capture device providers.
`LogMsgFileName (Get/Set)`	Get or set an optional filename for storing log messages generated by CaptureCore.
`CacheFolderName (Get/Set)`	Get or set an optional cache folder for storing any cached data files that CaptureCore uses for improving performance.
`GetMillisecondCount`	Returns the current time in milliseconds, used internally by CaptureCore.
`IdToString`	Converts an id enumeration value to its string representation.
`StringToId`	Converts a string representation returned by *IdToString* back to its id enumeration value.

3.2.1 Version

Version returns the version string of the CaptureCore assembly/framework file.

Syntax

.Net

C#	`string Version { get; }`
C++	`property System::String^ Version { System::String^ get(); }`
VB	`ReadOnly Property Version As String`

ObjC

```
- (NSString *) version
```

Return Value

A string containing the major, minor, revision, and build numbers for the CaptureCore assembly/framework. The string is a dot delimited string of numbers in the form of `mmm.nnn.rrr.bbb`, where `mmm` is the major version, `nnn` is the minor version, `rrr` is the interface revision, and `bbb` is the build number. The version string is static and doesn't change dynamically.

Remarks

The version number is used internally by Phase One for tracking the version of CaptureCore, and corresponds to the file version of the CaptureCore assembly/framework. The major,

minor and build numbers generally correspond to application or SDK releases. Only the interface revision number represents an actual iteration in the CaptureCore specification. Applications can use the *Revision* method to retrieve directly the interface revision as a number.

3.2.2 Revision

Revision returns the interface revision number of the CaptureCore assembly/framework file.

The interface revision number is directly related to the CaptureCore specification described by this document. This number can be used to ensure that the CaptureCore version being used corresponds to what the application was designed for.

Syntax

.Net

C#	`ushort Revision { get; }`
C++	`property System::UInt16 Revision { System::UInt16 get(); }`
VB	`ReadOnly Property Revision As UShort`

ObjC

```
- (uint16_t) revision
```

Return Value

The interface revision number of the CaptureCore assembly/framework file. The revision value is static and doesn't change dynamically.

3.2.3 Terminate

Terminate releases all resources used by CaptureCore objects. Calling this method is optional, and is called on the application's behalf when the application exits. Applications can use this method to explicitly control the release of CaptureCore resources. Once *Terminate* is called, CaptureCore can no longer be used, until the application is restarted.

Syntax

.Net

C#	`void Terminate()`
C++	`void Terminate()`
VB	`Sub Terminate`

ObjC

```
- (void) terminate
```

Remarks

Normally the resources used by any existing CaptureCore objects are automatically released when the objects no longer exist, that is when no references to those objects remain. A single instance of *ICaptureCore* persists even if it is no longer in use, and that instance may also hold references to *ICaptureProvider* objects, which likewise may hold references to *ICamera* objects, and so on. These persistent objects are only released when the application exits.

Alternatively, an application can call *Terminate* to release the resources held by these persistent objects when they are no longer needed. Some objects may continue to exist, but any significant resources held by them will be released. Any remaining objects will become unusable, and no part of CaptureCore can be used, before the application is restarted.

3.2.4 GetCaptureProviderList

GetCaptureProviderList returns a *ICaptureProviderList* object containing *ICaptureProvider* objects representing all supported capture device providers. An *ICaptureProvider* object could represent capture devices from different manufacturers, such as Phase One, or different protocols for devices from the same manufacturer.

Syntax

.Net

C#	`ICaptureProviderList  GetCaptureProviderList()`
C++	`ICaptureProviderList^ GetCaptureProviderList()`
VB	`Function GetCaptureProviderList As ICaptureProviderList`

ObjC

```
- (P1CaptureCore_CaptureProviderList *) getCaptureProviderList
```

Return Value

An *ICaptureProviderList* object containing all *ICaptureProvider* objects that are supported by this instance of *ICaptureCore*. A NULL reference or an empty list is returned if no providers are supported.

Remarks

All *ICaptureProvider* objects are generally created during startup, or during the first call to *GetCaptureCore* or *GetCaptureProviderList*. All providers that are successfully created and initialized are returned in the provider list. After initialization the list is static – providers will not be added or removed from the list.

3.2.5 LogMsgFileName (Get/Set)

LogMsgFileName get or sets an optional filename for storing log messages generated by CaptureCore.

Syntax

.Net

C#	`string GetLogMsgFileName()`
	`void SetLogMsgFileName( string fileName )`
C++	`System::String^ GetLogMsgFileName()`
	`void SetLogMsgFileName( System::String^ fileName )`
VB	`Function GetLogMsgFileName As String`
	`Sub SetLogMsgFileName( fileName As String )`

ObjC

```
- (NSString *) logMsgFileName
- (void) setLogMsgFileName: (NSString *) fileName
```

Parameters (Set)

fileName — Path to the log file that CaptureCore will write log messages to. An empty path or a NULL reference will clear the current file name, and disable the writing of log messages to a file.

Return Value (Get)

The path to the file that is currently set as the CaptureCore log file. A NULL reference or an empty path is returned if the writing of log messages is currently disabled.

Remarks

CaptureCore writes log messages to the indicated file in Unicode format, and always appends to the current file's contents. Note that the file size is not limited by CaptureCore and may grow quite large over a long period of time.

Log messages include the date, time and thread ID for each message. Log messages can be useful for tracking the internal operations carried out by CaptureCore, as well as logging errors and warnings.

3.2.6 CacheFolderName (Get/Set)

CacheFolderName gets or set an optional cache folder for storing any cached data files that CaptureCore uses for improving performance.

Syntax

.Net

C#	`string GetCacheFolderName()`
	`SetCacheFolderName( string folderName )`
C++	`System::String^ GetCacheFolderName()`
	`void SetCacheFolderName( System::String^ folderName )`
VB	`Function GetCacheFolderName As String`
	`Sub SetCacheFolderName( folderName As String )`

ObjC

`- (NSString *) cacheFolderName`
`- (void) setCacheFolderName: (NSString *) folderName`

Parameters (Set)

folderName	Path to the folder to store CaptureCore cache files in. An empty path or a NULL reference will clear the current folder name, and disable the storing of cache files.

Return Value (Get)

The path to the folder that is the current cache folder for CaptureCore. A NULL reference or an empty path is returned if storing cache files is currently disabled.

Remarks

CaptureCore requires write access to the cache folder. In addition to creating cache files, CaptureCore will create subfolders to organize the files. It is not necessary to specify a cache folder for the proper operation of CaptureCore.

Cache files store data that can improve performance, such as storing calibration data for a specific device so that it doesn't need to read the calibration data every time the device connects. Cache files may also contain capture settings for the device that are not persisted on the device.

3.2.7 GetMillisecondCount

GetMillisecondCount returns the current time in milliseconds, used internally by CaptureCore.

Syntax

.Net

C#	`uint GetMillisecondCount()`
C++	`System::UInt32 GetMillisecondCount ()`
VB	`Function GetMillisecondCount As UInteger`

ObjC

```
- (uint32_t) getMillisecondCount
```

Return Value

Time in milliseconds used by CaptureCore internally. The time returned is relative to a fixed point in time, and should not be used to determine the absolute time.

3.2.8 IdToString

IdToString converts an id enumeration value to its string representation.

Syntax

.Net

C#	`string IdToString( uint idValue )`
C++	`System::String^ IdToString( System::UInt32 idValue )`
VB	`Function IdToString( idValue As UInteger) As String`

ObjC

```
- (NSString *) idToString: (uint32_t) idValue
```

Parameters

idValue	The enumeration value of the id to convert to a string. Generally these are capability or property ids, documented in the capability and property reference sections.

Return Value

A string representation for the enumeration value given by *idValue*. A string representation is the name of the enumeration as a string. For example, the string "kCameraProperty_Model" is the string representation for the enumeration value *kCameraProperty_Model*.

If no string representation exists for *idValue*, the method returns an empty string or a NULL reference.

3.2.9 StringToId

StringToId converts a string representation returned by *IdToString* back to its id enumeration value.

Syntax

.Net

C#	`StringToId( string strId )`
C++	`void StringToId( System::String^ strId )`
VB	`Function StringToId( strId As String ) As UInteger`

ObjC

```
- (uint32_t) stringToId: (NSString *) strId
```

Parameters

strId	A string returned by *IdToString*, or a string representation of an id enumeration value, typically a capability or property id enumeration value. A string representation is just the name of the enumeration as a string. For example, the string "kCameraProperty_Model" is the string representation for the enumeration value *kCameraProperty_Model*.

Return Value

The id enumeration value corresponding to *strId*. If no enumeration value exists for *strId*, the method returns 0.

3.3 ICaptureProviderList (P1CaptureCore_CaptureProviderList)

The *ICaptureProviderList* class is a list container for *ICaptureProvider* objects. It is a child object of *ICaptureCore*, and inherits from *IChildObject* and *IObjectList*.

Members

Inherited from IChildObject	
`Parent`	Returns the parent *ICaptureCore* object of this object.
Inherited from IObjectList	
`Size`	Returns the number of *ICaptureProvider* items in the list.
`IsEmpty`	Returns true if the list is empty.
`First`	Returns a reference to the first *ICaptureProvider* item in the list.
`Last`	Returns a reference to the last *ICaptureProvider* item in the list.
`Next`	Returns a reference to the next *ICaptureProvider* item in the list following a specified *ICaptureProvider* item already in the list.
`Previous`	Returns a reference to the previous *ICaptureProvider* item in the list preceding a specified *ICaptureProvider* item already in the list.
`Insert`	Inserts a new item in front of another specified *ICaptureProvider* item in the list. Requires insert access rights.
`Remove`	Removes a specified *ICaptureProvider* item from the list. Requires remove access rights.
`Clear`	Removes all items from the list. Requires remove access rights.
`GetAccess`	Returns the access rights for this list as a bitmask of *EnumListAccess* values.
`HasAccess`	Returns true if the list allows the specified access rights.

3.4 ICaptureProvider (P1CaptureCore_CaptureProvider)

The *ICaptureProvider* class represents a supported capture device provider. It could represent capture devices from different manufacturers, such as Phase One, or different protocols for devices from the same manufacturer.

ICaptureProvider is a child object of *ICaptureCore*, and inherits from *IChildObject*, *ICaptureObject*, *IErrorSource*, and *IEventSource*. It is a parent to *ICamera* (*ICameraList*), *ICapability* (*ICapabilityList*), and *IProperty* (*IPropertyList*) objects.

Members

`IsAvailable`	Returns true if the *ICaptureProvider* object is currently available to be used. Unavailable objects throw an exception if any method other than this method is called.
`GetCameraList`	Returns a list of *ICamera* objects for the currently attached cameras for this provider.
`GetCamera`	Returns the *ICamera* object corresponding to a specified camera ID.
Inherited from IChildObject	
`Parent`	Returns the parent *ICaptureCore* object of this object.
Inherited from ICaptureObject	
`Id`	Returns an unique ID representing the *ICaptureProvider* object.
`GetCapabilityList`	Returns a reference to a *ICapabilityList* object containing all *ICapability* objects for this *ICaptureProvider* object.
`GetPropertyList`	Returns a reference to a *IPropertyList* object containing all *IProperty* objects for this *ICaptureProvider* object.
`GetCapability`	Returns a reference to an *ICapability* object for this *ICaptureProvider* object, with a specified capability ID.
`GetProperty`	Returns a reference to an *IProperty* object for this *ICaptureProvider* object, with a specified property ID.
Inherited from IErrorSource	
`GetError`	Returns the next *IErrorObject* object, if any, for the *ICaptureProvider* object.
Inherited from IEventSource	
`AddReceiver`	Attaches an *IEventReceiver* object to receive events (*IEventObject*) from the *ICaptureProvider* object.
`RemoveReceiver`	Detaches a previously attached *IEventReceiver* object so that it no longer receives events (*IEventObject*) from the *ICaptureProvider* object.

Events

General `(EnumCaptureProviderEventId)`	
`kCaptureProviderEvent_CameraAdded`	A camera object has been added to the camera list.
`kCaptureProviderEvent_CameraRemoved`	A camera object has been removed from the camera list.

Inherited from ICaptureObject (EnumCaptureObjectEventId)	
kCaptureObjectEvent_CapabilityChange	A capability's value has changed.
kCaptureObjectEvent_PropertyChange	A property's value has changed.
kCaptureObjectEvent_SettingDescriptorChange	A property's setting descriptor has changed.
Inherited from IErrorSource (EnumGeneralEventId)	
kEventId_Error	An error has occurred on a background thread. Indicates that a new *IErrorObject* object has been queued by this object.
Inherited from IEventSource (EnumGeneralEventId)	
kEventId_All	Used for subscribing or unsubscribing to all events via *AddReceiver* or *RemoveReceiver*.

3.4.1 IsAvailable

IsAvailable returns true if the *ICaptureProvider* object is currently available to be used. Even if a capture provider object is created and initialized it may still not be currently available. Some capture device types may only be available to one application at a time, or may require certain files or services to be installed. In these cases *IsAvailable* will return false.

Syntax

.Net

C#	`bool IsAvailable()`
C++	`bool IsAvailable()`
VB	`Function IsAvailable As Boolean`

ObjC

```
- (BOOL) isAvailable
```

Return Value

True if the *ICaptureProvider* object is currently available for use.

Remarks

Unavailable objects throw an exception if any method other than *IsAvailable* is called.

The return value of *IsAvailable* is constant for any instance of an *ICaptureProvider* object, so *IsAvailable* may be checked just once for each *ICaptureProvider*. The availability of each *ICaptureProvider* object is determined during CaptureCore initialization. Thus if a provider is not available, and the conditions preventing its availability are removed, CaptureCore must be unloaded and restarted before *IsAvailable* will return true.

3.4.2 GetCameraList

GetCameraList returns a list of *ICamera* objects for the currently attached cameras supported by this provider.

Syntax

.Net

C#	`ICameraList  GetCameraList()`
C++	`ICameraList^ GetCameraList()`
VB	`Function GetCameraList As ICameraList`

ObjC

```
- (P1CaptureCore_CameraList *) getCameraList
```

Return Value

An *ICameraList* object containing an *ICamera* object for each currently attached camera that is supported by this *ICaptureProvider*. A NULL reference or an empty list is returned if no cameras are connected.

Remarks

The *ICameraList* object returned is a copy of an internal camera list. This internal camera list is dynamic and can change as cameras are added or removed. When the internal list changes, a *kCaptureProviderEvent_CameraAdded* or *kCaptureProviderEvent_CameraRemoved* event is sent by the *ICaptureProvider*, informing the application that a new instance of the *ICameraList* object is available.

3.4.3 GetCamera

GetCamera returns an *ICamera* object corresponding to a specified camera ID in the camera list of this *ICaptureProvider*.

Syntax

.Net

C#	`ICamera  GetCamera( uint cameraID )`
C++	`ICamera^ GetCamera( System::UInt32 cameraID )`
VB	`Function GetCamera( cameraID As UInteger ) As ICamera`

ObjC

```
- (P1CaptureCore_Camera *) getCamera: (uint32_t) cameraID
```

Parameters

cameraID — Numerical id of the *ICamera* object to return. The Id corresponds to the value returned by the *ICamera* Id member.

Return Value

The *ICamera* object corresponding to the *cameraID* parameter. If no matching camera object is found, then a NULL reference is returned.

3.4.4 kCaptureProviderEvent_CameraAdded

This event is posted by the *ICaptureProvider* object when a new *ICamera* object is added to the camera list of the *ICaptureProvider* object.

Arguments

0 — [Optional] The camera ID of the *ICamera* object that was added to the camera list. This argument may not always be present.

3.4.5 kCaptureProviderEvent_CameraRemoved

This event is posted by the *ICaptureProvider* object when an existing *ICamera* object is removed from the camera list of the *ICaptureProvider* object.

Arguments

0 — [Optional] The camera ID of the *ICamera* object that was removed from the camera list. This argument may not always be present.

3.5 ICameraList (P1CaptureCore_CameraList)

The *ICameraList* class is a list container for *ICamera* objects. It is a child object of *ICaptureProvider*, and inherits from *IChildObject* and *IObjectList*.

The *ICameraList* object returned by the *ICaptureProvider* method *GetCameraList* is a copy of an internally maintained camera list. This internal list can change dynamically in response to method calls or events, such as a camera being connected or disconnected. Since the returned list is a copy it doesn't change dynamically. This protects the application from issues that can arise when iterating through a list that it also changing during the iteration. The *ICaptureProvider* object sends an event when its internal camera list changes, thus allowing the application to retrieve a new copy if desired.

Members

`GetCamera`	Returns the *ICamera* object corresponding to a specified camera ID.
Inherited from IChildObject	
`Parent`	Returns the parent *ICaptureProvider* object of this object.
Inherited from IObjectList	
`Size`	Returns the number of *ICamera* items in the list.
`IsEmpty`	Returns true if the list is empty.
`First`	Returns a reference to the first *ICamera* item in the list.
`Last`	Returns a reference to the last *ICamera* item in the list.
`Next`	Returns a reference to the next *ICamera* item in the list following a specified *ICamera* item already in the list.
`Previous`	Returns a reference to the previous *ICamera* item in the list preceding a specified *ICamera* item already in the list.
`Insert`	Inserts a new item in front of another specified *ICamera* item in the list. Requires insert access rights.
`Remove`	Removes a specified *ICamera* item from the list. Requires remove access rights.
`Clear`	Removes all items from the list. Requires remove access rights.
`GetAccess`	Returns the access rights for this list as a bitmask of *EnumListAccess* values.
`HasAccess`	Returns true if the list allows the specified access rights.

3.5.1 GetCamera

GetCamera returns an *ICamera* object corresponding to a specified camera ID in the camera list.

Syntax

.Net

C#	`ICamera  GetCamera( uint cameraID )`
C++	`ICamera^ GetCamera( System::UInt32 cameraID )`
VB	`Function GetCamera( cameraID As UInteger ) As ICamera`

ObjC

```
- (P1CaptureCore_Camera *) getCamera: (uint32_t) cameraID
```

Parameters

cameraID	Numerical id of the *ICamera* object to return. The Id corresponds to the value returned by the *ICamera* Id member.

Return Value

The *ICamera* object corresponding to the *cameraID* parameter. If no matching camera object is found, then a NULL reference is returned.

3.6 ICamera (P1CaptureCore_Camera)

The *ICamera* class represents an attached capture device and provides capture setup and control methods.

ICamera is a child object of *ICaptureProvider*, and inherits from *IChildObject*, *ICaptureObject*, *IErrorSource*, *IEventSource*, and *IProgressSource*. It is a parent to *ICaptureImage* (*ICaptureImageList*), *ICapability* (*ICapabilityList*), *IProperty* (*IPropertyList*), and *IProgressStatus* objects.

If the *ICamera* object has the *kCameraProperty_HostStorageCapacity* property, then the application will need to set the property to indicate the amount of disk space that is available for storage of images. Such devices generally disable capture, when the amount of host storage space is insufficient to store any further images.

The *ICamera* class and is the fundamental class in CaptureCore. *ICamera* objects are used for communicating and controlling a capture device, and for capturing and transferring images from it.

ICamera objects can be in various states. When an *ICamera* object is first retrieved from its parent *ICaptureProvider* object, it is not yet open. The device may in fact not be available, if it is in use by another application (see *IsAvailable*). The application cannot communicate with the device before calling *Open*. Once *Open* is called, an application can get and set properties on the device, and begin image capture.

To begin capturing, the application must first call *StartCapture*. The device will automatically capture and transfer images to the host, whenever the shutter release is pressed physically on the device or when the *ShutterRelease* method is called. To stop capturing and release all capture related resources, the application calls the *StopCapture* method. Capturing will also stop if an error occurs. Finally, the application can also pause capturing, which temporarily disables the shutter and optionally pauses image transfer without doing a full stop, by calling the *PauseCapture* method. The events *kCameraEvent_CapturingStarted* and *kCameraEvent_CapturingStopped* are posted by the *ICamera* object when its capturing state changes.

Captured images are placed in a *ICaptureImageList* object and can be retrieved by the *GetNextCaptureImage* or *GetCaptureImageQueue* methods. During capture and transfer images can be queued in the capture device, or be in transfer. Images that are not yet in the capture image queue are called pending images, and the number of pending images is returned by the *PendingImageCount* method. The *kCameraEvent_PendingImageCount* change event is posted when the number of pending images changes.

At any point in time, the capture device can be physically removed from the host. When this occurs the *IsConnected* method returns false, and a *kCameraEvent_CameraDisconnected* event is posted. After a device is disconnected, it may no longer be used an only the methods *StopCapture* and *Close* can be called. If the same device is reconnected, a new *ICamera* object is created to represent the device.

Members

`IsAvailable`	Returns true if the *ICamera* object is available for use.
`Open`	Opens the *ICamera* object and initializes communication with the attached device. Allows access to other members.

`Close`	Closes the *ICamera* object, disconnecting from the attached device, and releasing all resources.
`IsOpen`	Returns true if the *ICamera* object is open, that is if a previous call to *Open* has succeeded, and *Close* has not yet been called.
`IsConnected`	Returns true if the device associated with the *ICamera* object is still connected to the host computer.
`StartCapture`	Enables the device to capture images, and starts the transfer of captured images from the device to the host computer.
`PauseCapture`	If capturing is started, disables the capturing of images and optionally their transfer, without releasing all resources for capturing. Call *StartCapture* again to continue the capture and transfer of images.
`StopCapture`	If capturing is started, disables the capturing of images, optionally transfers any outstanding images, and finally releases all resources related to capturing.
`IsCapturing`	Returns true if the *ICamera* object is ready to capture and transfer images, following a call to *StartCapture*.
`IsCapturingPaused`	Returns true if the *ICamera* object is ready to capture images, but is currently paused following a call to *PauseCapture*.
`PendingImageCount`	Returns the number of images in the capture device's internal buffer or being transferred to the captured image queue.
`ShutterRelease`	Requests that the capture device capture an image.
`GetNextCaptureImage`	Retrieves the next captured image from the captured image queue. The returned image is removed from the queue.
`GetCaptureImageQueue`	Returns the captured image queue as an *ICaptureImageList* object.
`MaxCaptureQueueSize (Get/Set)`	Gets or sets the maximum captured image queue size.
Inherited from IChildObject	
`Parent`	Returns the parent *ICaptureProvider* object of this object.
Inherited from ICaptureObject	
`Id`	Returns an unique ID representing the *ICamera* object.
`GetCapabilityList`	Returns a reference to a *ICapabilityList* object containing all *ICapability* objects for this *ICamera* object.
`GetPropertyList`	Returns a reference to a *IPropertyList* object containing all *IProperty* objects for this *ICamera* object.
`GetCapability`	Returns a reference to an *ICapability* object for this *ICamera* object, with a specified ID.
`GetProperty`	Returns a reference to an *IProperty* object for this *ICamera* object, with a specified ID.

Inherited from IErrorSource	
`GetError`	Returns the next *IErrorObject* object, if any, for the *ICamera* object.
Inherited from IEventSource	
`AddReceiver`	Attaches an *IEventReceiver* object to receive events (*IEventObject*) from the *ICamera* object.
`RemoveReceiver`	Detaches a previously attached *IEventReceiver* object so that it no longer receives events (*IEventObject*) from the *ICamera* object.
Inherited from IProgressSource	
`GetProgress`	Returns the next *IProgressStatus* object in the progress queue for this *ICamera* object.

Events

General `(EnumCameraEventId)`	
`kCameraEvent_CameraDisconnected`	The device associated with the camera has been disconnected.
`kCameraEvent_ImageReceived`	A new image object has been added to the image list.
`kCameraEvent_PendingImageCountChange`	The pending image count has changed.
`kCameraEvent_CapturingStarted`	Image capture has been started.
`kCameraEvent_CapturingStopped`	Image capture has been stopped.
Inherited from ICaptureObject `(EnumCaptureObjectEventId)`	
`kCaptureObjectEvent_CapabilityChange`	A capability's value has changed.
`kCaptureObjectEvent_PropertyChange`	A property's value has changed.
`kCaptureObjectEvent_SettingDescriptorChange`	A property's setting descriptor has changed.
Inherited from IErrorSource `(EnumGeneralEventId)`	
`kEventId_Error`	An error has occurred on a background thread. Indicates that a new *IErrorObject* object has been queued by this object.
Inherited from IEventSource `(EnumGeneralEventId)`	
`kEventId_All`	Used for subscribing or unsubscribing to all events via *AddReceiver* or *RemoveReceiver*.
Inherited from IProgressSource `(EnumGeneralEventId)`	
`kEventId_ProgressUpdate`	Indicates that a new *IProgressStatus* object has been queued by this object.
Phase One device specific `(EnumPhaseOneCameraEventId)`	
`kP1CameraEvent_MacCreateLocalIsochPortError`	Mac OS only. There is insufficient memory below the 2GB memory boundary for the operating system to setup an isochronous FireWire transfer port between the host and the device.

Progress Status

General `(EnumCameraProgressId)`	
`kCameraProgress_Open`	Progress status for the Open method.
`kCameraProgress_ImageTransfer`	Progress status for image transfers.

3.6.1 IsAvailable

IsAvailable returns true if the *ICamera* object is currently available to be used. Even if a camera object is created and initialized it may still not be currently available. For example, some capture devices may only be available to one application at a time. In these cases *IsAvailable* will return false.

Syntax

.Net

C#	`bool IsAvailable()`
C++	`bool IsAvailable()`
VB	`Function IsAvailable As Boolean`

ObjC

```
- (BOOL) isAvailable
```

Return Value

True if the *ICamera* object is currently available for use.

Remarks

Unavailable objects throw an exception if any method other than *IsAvailable* is called.

Following a successful call to *Open*, the *ICamera* object will remain available for use by the process that called *Open*. *IsAvailable* will return true as long as the *ICamera* object remains opened (*IsOpen* returns true) and connected (*IsConnected* returns true). If an *ICamera* object is already open in another process, *IsAvailable* will generally return false, and no other methods including *Open* can be called on the object.

IsAvailable returns false if the associated device is disconnected (*IsConnected* returns true), even if the *ICamera* object is currently open.

3.6.2 Open

Open opens the *ICamera* object and initializes communication with the attached device. This reserves the device for use by the application.

Open must be called before many of the other class members, such as *StartCapture*, *PauseCapture*, *StopCapture*, *ShutterRelease* and so on. It is generally not necessary to call Open for accessing inherited members like *GetCapability* and *GetProperty*. However, the return result from calling some methods may differ depending upon whether the device is open or not. See the documentation for other *ICamera* members for whether they require the object to be open.

Syntax

.Net

C#	`void Open()`
C++	`void Open()`
VB	`Sub Open`

ObjC

```
- (void) open
```

Remarks

Once open, the *ICamera* object remains open until explicitly closed by calling *Close*, or the device is no longer connected and there are no remaining references to the *ICamera* object.

The application should avoid leaving an *ICamera* object open, once it no longer needs it open.

3.6.3 Close

Close closes the *ICamera* object, disconnecting from the attached device, and releasing all resources. *Close* is called after a prior call to *Open*, when the application no longer needs to reserve the device associated with the *ICamera* object.

Close can be called on an already closed device, though this does nothing.

Syntax

.Net

C#	`void Close()`
C++	`void Close()`
VB	`Sub Close`

ObjC

```
- (void) close
```

Remarks

Close automatically calls *StopCapture* with *bWaitOnPending* set to false, which may discard any pending images not yet transferred from the device. If the application wishes to transfer any pending images, it should call *StopCapture* with *bWaitOnPending* set to true, prior to calling *Close*. Images already transferred to the *ICamera* object are not discarded by calling *Close*.

3.6.4 IsOpen

IsOpen returns true if the *ICamera* object is open, that is if a previous call to *Open* has succeeded, and *Close* has not yet been called.

Syntax

.Net

C#	`bool IsOpen()`
C++	`bool IsOpen()`
VB	`Function IsOpen As Boolean`

ObjC

```
- (BOOL) isOpen
```

Return Value

True if the *ICamera* object is currently in an open state, otherwise false.

3.6.5 IsConnected

IsConnected returns true if the device associated with the *ICamera* object is currently connected to the host computer.

Syntax

.Net

C#	`bool IsConnected()`
C++	`bool IsConnected()`
VB	`Functions IsConnected As Boolean`

ObjC

```
- (BOOL) isConnected
```

Return Value

True if the device associated with the *ICamera* object is currently connected to the host computer.

Remarks

The return value of *IsConnected* can change in response to external events. So multiple calls to *IsConnected* in a row could return different values. When the device is disconnected, a *kCameraEvent_CameraDisconnected* event is sent.

Once a device is disconnected, that is *IsConnected* returns false, the state of the *ICamera* object will not return to connected. If the device that was formerly connected is reconnected, a new *ICamera* object is created instead.

An *ICamera* object remains open, even if the device is no longer connected. An application must still explicitly call Close, when it no longer wishes to communicate with the device. Usually, an application calls Close immediately in response to an open device being disconnected.

IsAvailable will also return false, once *IsConnected* returns false.

3.6.6 StartCapture

StartCapture enables the device to capture images, and starts the transfer of captured images from the device to the host computer. *StartCapture* can also be called following a call to *PauseCapture* to restart the capture and transfer of images.

The *ICamera* object must be in an open state (that is a successful call to *Open* has been made) before calling *StartCapture*. *StartCapture* can be called on a device that is already capturing, though this does nothing, unless capturing is paused.

StartCapture allocates any resources that are necessary for the capture and transfer of images, such as internal image buffers. These resources may be significant and are usually released when *StopCapture* is called, but may also only be released when *Close* is called.

Syntax

.Net

C#	`void StartCapture()`
C++	`void StartCapture()`
VB	`Sub StartCapture`

ObjC

```
- (void) startCapture
```

Remarks

When the capturing state changes to started, a *kCameraEvent_CapturingStarted* event is sent.

This method is only supported if the *ICamera* object has the *kCameraCapability_Capture* capability with a value of true.

3.6.7 PauseCapture

If capturing is started, *PauseCapture* disables the capturing of images and optionally their transfer, without releasing all resources for capturing. It disables the shutter of the device, and

optionally also disables the transfer of pending images. Call *StartCapture* again to continue the capture and transfer of images, or *Stop* to stop capturing and release all capturing resources.

The *ICamera* object must be in a capturing state (that is a successful call to *StartCapture* has been made) before calling *PauseCapture*. *PauseCapture* can be called when the *ICamera* object is already paused. Doing so can change whether image transfer is paused, but the capture of images will remain disabled.

Syntax

.Net

C#	`void PauseCapture( bool bPauseTransfer )`
C++	`void PauseCapture( bool bPauseTransfer )`
VB	`Sub PauseCapture( bPauseTransfer As Boolean )`

ObjC

```
- (void) pauseCapture: (BOOL) pauseTransfer
```

Parameters

bPauseTransfer — If true, both the capture and transfer of images is paused. If false, only the capture of images is paused, and pending images continue to transfer.

Remarks

This method is only supported if the *ICamera* object has the *kCameraCapability_Capture* and the *kCameraCapability_PauseCapture* capabilities with a value of true. In addition, the *bPauseTransfer* parameter is only supported if the *ICamera* object also has the *kCameraCapability_PauseCaptureAndTransfer* capability with the value of true.

3.6.8 StopCapture

If capturing is started, *StopCapture* disables the capturing of images, optionally transfers any outstanding images, and finally releases all resources related to capturing. *StopCapture* disables the shutter of the device and generally releases any resources allocated by *StartCapture*, though in some cases some resources may only be released by a call to *Close*.

The *ICamera* object must be in an open state (that is a successful call to *Open* has been made) before calling *StopCapture*. *StopCapture* can be called on a device that is not capturing, though this does nothing.

StopCapture can optionally wait on pending images. If the *bWaitOnPending* parameter is true, then *StopCapture* disables capture and blocks until all pending images are transferred. This can take some time. If *bWaitOnPending* is false, pending images are generally discarded.

Syntax

.Net

C#	`void StopCapture( bool bWaitOnPending )`
C++	`void StopCapture( bool bWaitOnPending )`
VB	`Sub StopCapture( bWaitOnPending As Boolean )`

ObjC

```
- (void) stopCapture: (BOOL) waitOnPending
```

Parameters

bWaitOnPending	If true, *StopCapture* blocks until all pending images are transferred. If false, pending images are generally discarded.

Remarks

The capturing state can also be automatically stopped by an external event, such as an error or if the device is disconnected. When the capturing state is stopped, a *kCameraEvent_CapturingStopped* event is sent.

If *bWaitOnPending* is true, *StopCapture* blocks until all pending images are transferred. If there are any conditions that are hindering the transfer of pending images, then *StopCapture* may block for a very long time, until these conditions are no longer present. Generally, the transfer of images requires room in the image queue as well as memory for the new images. Thus the application should continue to process pending images as they arrive, even during a call to *StopCapture*. If the application stops processing transferred images, for example when there is insufficient storage capacity, this could also result in *StopCapture* blocking until the application begins processing images again.

If *StopCapture* is blocking while transferring pending images, then most other *ICamera* methods, such as *Open*, *Close*, *StartCapture*, and so on, will also block until *StopCapture* is complete.

This method is only supported if the *ICamera* object has the *kCameraCapability_Capture* capability with a value of true.

3.6.9 IsCapturing

IsCapturing returns true if the *ICamera* object is ready to capture and transfer images, following a call to *StartCapture*. It returns the *ICamera* object's current capturing state. Even if capturing is paused, *IsCapturing* still returns true.

Syntax

.Net

C#	`bool IsCapturing()`
C++	`bool IsCapturing()`
VB	`Function IsCapturing As Boolean`

ObjC

```
- (BOOL) isCapturing
```

Return Value

True, if the *ICamera* object's is ready to capture and transfer images, even if it is paused.

Remarks

The capturing state of an *ICamera* object can be automatically stopped without an explicit call to *StopCapture*, for example in the event of an error or if the device is disconnected. When the capturing state changes, a *kCameraEvent_CapturingStarted* or *kCameraEvent_CapturingStopped* event is sent.

This method is only supported if the *ICamera* object has the *kCameraCapability_Capture* capability with a value of true.

3.6.10 IsCapturingPaused

IsCapturingPaused returns true if the *ICamera* object is ready to capture images, but is currently paused following a call to *PauseCapture*.

Syntax

.Net

C#	`bool IsCapturingPaused()`
C++	`bool IsCapturingPaused()`
VB	`Function IsCapturingPaused As Boolean`

ObjC

```
- (BOOL) isCapturingPaused
```

Return Value

True, if the *ICamera* object is ready to capture images, but is currently paused.

Remarks

The capturing state is started but paused, when *StartCapture* has been called to start capturing, followed by a *PauseCapture* call. If capturing is paused, capturing can be restarted by calling *StartCapture*, or stopped by calling *StopCapture*.

This method is only supported if the *ICamera* object has the *kCameraCapability_Capture* capability with a value of true.

3.6.11 PendingImageCount

PendingImageCount returns the number of images in the capture device's internal buffer or being transferred to the captured image queue. Images already added to the captured image queue are not counted as pending.

Syntax

.Net

C#	`uint PendingImageCount()`
C++	`System::UInt32 PendingImageCount()`
VB	`Function PendingImageCount As UInteger`

ObjC

```
- (uint32_t) pendingImageCount
```

Return Value

The number of pending images either in the device or in transfer. If the return value is 0xFFFFFFFF (i.e. –1), then there are pending images but the actual number of images is unknown.

Remarks

When the pending image count changes, the *kCameraEvent_PendingImageCountChange* event is sent.

This method is only supported if the *ICamera* object has the *kCameraCapability_Capture* and *kCameraCapability_PendingImageCount* capabilities with a value of true.

3.6.12 ShutterRelease

ShutterRelease requests that the capture device capture an image. The device will generally capture an image if it is possible to do so. However if the device is busy, cannot focus the lens, or encounters some error, then no image may actually be captured. *ShutterRelease* does not block while the image is being captured, it only sends a capture request to the device.

The *ICamera* object must be in a capturing state and not paused (that is a successful call to *StartCapture* must have been made) before calling *ShutterRelease*.

Syntax

.Net

C#	`void ShutterRelease()`
C++	`void ShutterRelease()`
VB	`Sub ShutterRelease`

ObjC

```
- (void) shutterRelease
```

Remarks

This method is only supported if the *ICamera* object has the *kCameraCapability_Capture* and *kCameraCapability_ShutterRelease* capabilities with a value of true.

3.6.13 GetNextCaptureImage

GetNextCaptureImage retrieves the next captured image from the captured image queue. The returned image is removed from the queue.

Syntax

.Net

C#	`ICaptureImage  GetNextCaptureImage()`
C++	`ICaptureImage^ GetNextCaptureImage()`
VB	`Function GetNextCaptureImage As ICaptureImage`

ObjC

```
- (P1CaptureCore_CaptureImage *) getNextCaptureImage
```

Return Value

The next *ICaptureImage* object in the captured image queue. A NULL reference is returned if there are no more captured images in the queue.

Remarks

Captured images remain in the captured image queue until they are removed by the application or the *ICamera* object is destroyed. Calling the *ICamera* method *StopCapture* or *Close* does not discard any captured images. If images are not removed from the queue, the amount of memory used can grow significantly. To release memory used by an image, the *ICaptureImage* object must both be removed from the queue, and all references to the object released. Alternatively, the *ICaptureImage* method *Close* can be called on an *ICaptureImage* object to release the memory used by the image.

This method is equivalent to calling *GetCaptureImageQueue* and calling the *ICaptureImageList* methods *First* and *Remove*, while checking all calls for NULL references.

This method is only supported if the *ICamera* object has the *kCameraCapability_Capture* capability with a value of true.

3.6.14 GetCaptureImageQueue

GetCaptureImageQueue returns the captured image queue as an *ICaptureImageList* object. The captured image queue contains all successfully captured and transferred images.

Syntax

.Net

C#	`ICaptureImageList  GetCaptureImageQueue()`
C++	`ICaptureImageList^ GetCaptureImageQueue()`
VB	`Function GetCaptureImageQueue As ICaptureImageList`

ObjC

```
- (P1CaptureCore_CaptureImageList *) getCaptureImageQueue
```

Return Value

An *ICaptureImageList* object containing an *ICaptureImage* object for each captured image. A NULL reference or an empty list is returned if there are no captured images.

Remarks

The returned *ICaptureImageList* object is not a copy of an internal queue. Thus the contents of the *ICaptureImageList* object can change dynamically as images are added or removed.

Captured images remain in the captured image queue until they are removed by the application or the *ICamera* object is destroyed. Calling the *ICamera* method *StopCapture* or *Close* does not discard any captured images. If images are not removed from the queue, the amount of memory used can grow significantly. To release memory used by an image, the *ICaptureImage* object must both be removed from the queue, and all references to the object released. Alternatively, the *ICaptureImage* method *Close* can be called on an *ICaptureImage* object to release the memory used by the image.

This method is only supported if the *ICamera* object has the *kCameraCapability_Capture* capability with a value of true.

3.6.15 MaxCaptureQueueSize (Get/Set)

MaxCaptureQueueSize gets or sets the maximum captured image queue size. This limits the number of images that will be placed in the captured image queue. Whenever the queue is full, further transfer of images will be automatically disabled, and re-enabled again once there is space in the image queue.

Syntax

.Net

C#	`uint GetMaxCaptureQueueSize()`
	`void SetMaxCaptureQueueSize( uint max )`
C++	`System::UInt32 GetMaxCaptureQueueSize()`
	`void SetMaxCaptureQueueSize( System::UInt32 max )`
VB	`Function GetMaxCaptureQueueSize As UInteger`
	`Sub SetMaxCaptureQueueSize( max As UInteger )`

ObjC

```
- (uint32_t) maxCaptureQueueSize
- (void) setMaxCaptureQueueSize: (uint32_t) max
```

Parameters

max	The maximum number of captured images to store in the captured image queue. No limit is imposed, if *max* is set to 0 or 0xFFFFFFFF (i.e. –1).

Return Value

The current maximum captured image queue size. Zero or 0xFFFFFFFF (i.e. –1) is returned if there is no limit to the image queue.

Remarks

This method is only supported if the *ICamera* object has the *kCameraCapability_Capture* and *kCameraCapability_MaxCaptureQueueSize* capabilities with a value of true.

3.6.16 kCameraEvent_CameraDisconnected

This event is posted by the *ICamera* object when the device associated with the camera object has been disconnected.

Arguments

None

3.6.17 kCameraEvent_ImageReceived

This event is posted by the *ICamera* object when a new *ICaptureImage* object has been added to the image queue of the *ICamera* object.

Arguments

None

3.6.18 kCameraEvent_PendingImageCountChange

This event is posted by the *ICamera* object when the pending image count has changed.

Arguments

None

3.6.19 kCameraEvent_CapturingStarted

This event is posted by the *ICamera* object when image capturing has been started, following a call to *StartCapture*. Subsequent calls to *StartCapture* before a call to *StopCapture* will not generate another event.

Arguments

None

3.6.20 kCameraEvent_CapturingStopped

This event is posted by the *ICamera* object when the state of image capturing has changed to stopped, usually as a result of a call to *StopCapture*. Capturing may also stop because of an error.

Arguments

None

3.7 ICaptureImageList (P1CaptureCore_CaptureImageList)

The *ICaptureImageList* class is a list container for *ICaptureImage* objects. It is a child object of *ICamera*, and inherits from *IChildObject* and *IObjectList*.

The *ICaptureImageList* object returned by the *ICamera* method *GetCaptureImageQueue* can change dynamically in response to method calls or events, such as a new image being captured. Care should be taken to check for NULL return values when iterating the list, since it can change dynamically. New images are added instantaneously to the end of the list by a CaptureCore background thread. *ICaptureImageList* is still thread-safe like all objects in CaptureCore, but an application should be aware that the contents of the list can change at any moment.

Members

`GetCaptureImage`	Returns the *ICaptureImage* object corresponding to a specified image ID.
Inherited from IChildObject	
`Parent`	Returns the parent *ICamera* object of this object.
Inherited from IObjectList	
`Size`	Returns the number of *ICaptureImage* items in the list.
`IsEmpty`	Returns true if the list is empty.
`First`	Returns a reference to the first *ICaptureImage* item in the list.
`Last`	Returns a reference to the last *ICaptureImage* item in the list.
`Next`	Returns a reference to the next *ICaptureImage* item in the list following a specified *ICaptureImage* item already in the list.
`Previous`	Returns a reference to the previous *ICaptureImage* item in the list preceding a specified *ICaptureImage* item already in the list.
`Insert`	Inserts a new item in front of another specified *ICaptureImage* item in the list. Requires insert access rights.
`Remove`	Removes a specified *ICaptureImage* item from the list. Requires remove access rights.
`Clear`	Removes all items from the list. Requires remove access rights.
`GetAccess`	Returns the access rights for this list as a bitmask of *EnumListAccess* values.
`HasAccess`	Returns true if the list allows the specified access rights.

3.7.1 GetCaptureImage

GetCaptureImage returns an *ICaptureImage* object corresponding to a specified image ID in the image list.

Syntax

.Net

C#	`ICaptureImage  GetCaptureImage( uint imageID )`
C++	`ICaptureImage^ GetCaptureImage( System::UInt32 imageID )`

VB	`Function GetCaptureImage( imageID As UInteger ) As ICaptureImage`

ObjC

```
- (P1CaptureCore_CaptureImage *) getCaptureImage: (uint32_t) imageID
```

Parameters

imageID — Numerical id of the *ICaptureImage* object to return. The Id corresponds to the value returned by the *ICaptureImage* Id member.

Return Value

The *ICaptureImage* corresponding to the *imageID* parameter. If no matching image object is found, then a NULL reference is returned.

3.8 ICaptureImage (P1CaptureCore_CaptureImage)

The *ICaptureImage* class represents a captured image file, it contains the image file and provides methods for copying or saving the image file and accessing its metadata. Captured images are encapsulated in an image file format, that is specific for the capture device, such as TIFF or JPG. *ICaptureImage* methods do not give direct access to the image data, but provide access to the image file which encapsulates the image.

Even though *ICaptureImage* represents an image file, the file is generally only stored in memory. Captured image files can be quite large and thus can use a large amount of memory. This memory is released when the *ICaptureImage* object is destroyed (no longer used) or by calling the *Close* method. In some development environments, memory is automatically garbage collected, but the algorithm only releases memory when no more memory is available. In these environments, it is recommended to always call *Close* when the *ICaptureImage* object is no longer needed, to force the memory to be released immediately.

ICaptureImage objects are created and queued by the *ICamera* class. When the object is queued, a *kCameraEvent_ImageReceived* event is posted by *ICamera*. *ICaptureImage* objects can be retrieved by calling the *ICamera* method *GetNextCaptureImage*, or via the *ICaptureImageList* object returned from the *ICamera* method *GetCaptureImageQueue*.

ICaptureImage is a child object of *ICamera*, and inherits from *IChildObject*, *ICaptureObject*, *IErrorSource*, *IEventSource*. It is a parent to *ICapability* (*ICapabilityList*), and *IProperty* (*IPropertyList*) objects.

Members

`Close`	Closes the *ICaptureImage* object and releases all significant resources.
`FileSize`	Returns the size of the image file containing the captured image.
`SaveToFile`	Saves the image file to a specified filename path.
`SaveToBuffer`	Saves the image file to a specified memory buffer.
`GetImageData`	Returns an *IImageData* object giving access to the image data of the captured image.
`GetThumbnail`	Returns an *ICaptureImageThumbnail* object representing a thumbnail image of the captured image.
Inherited from IChildObject	
`Parent`	Returns the parent *ICamera* object of this object.
Inherited from ICaptureObject	
`Id`	Returns an unique ID representing the *ICaptureImage* object.
`GetCapabilityList`	Returns a reference to a *ICapabilityList* object containing all *ICapability* objects for this *ICaptureImage* object.
`GetPropertyList`	Returns a reference to a *IPropertyList* object containing all *IProperty* objects for this *ICaptureImage* object.
`GetCapability`	Returns a reference to an *ICapability* object for this *ICaptureImage* object, with a specified ID.

`GetProperty`	Returns a reference to an *IProperty* object for this *ICaptureImage* object, with a specified ID.
Inherited from IErrorSource	
`GetError`	Returns the next *IErrorObject* object, if any, for the *ICaptureImage* object.
Inherited from IEventSource	
`AddReceiver`	Attaches an *IEventReceiver* object to receive events (*IEventObject*) from the *ICaptureImage* object.
`RemoveReceiver`	Detaches a previously attached *IEventReceiver* object so that it no longer receives events (*IEventObject*) from the *ICaptureImage* object.

Events

Inherited from ICaptureObject `(EnumCaptureObjectEventId)`	
`kCaptureObjectEvent_CapabilityChange`	A capability's value has changed.
`kCaptureObjectEvent_PropertyChange`	A property's value has changed.
`kCaptureObjectEvent_SettingDescriptorChange`	A property's setting descriptor has changed.
Inherited from IErrorSource `(EnumGeneralEventId)`	
`kEventId_Error`	An error has occurred on a background thread. Indicates that a new *IErrorObject* object has been queued by this object.
Inherited from IEventSource `(EnumGeneralEventId)`	
`kEventId_All`	Used for subscribing or unsubscribing to all events via *AddReceiver* or *RemoveReceiver*.

3.8.1 Close

Close closes the *ICaptureImage* object and releases all significant resources. The memory used by an *ICaptureImage* object can be quite large since the captured image file is generally stored in memory. *Close* can be called when the *ICaptureImage* object is no longer needed to immediately release this memory. If Close is not called it will still be released automatically when the object is destroyed.

Syntax

.Net

C#	`void Close()`
C++	`void Close()`
VB	`Sub Close`

ObjC

```
- (void) close
```

Remarks

Memory used by *ICaptureImage* is released when the object is destroyed (no longer used) or by calling *Close*. In some development environments, memory is automatically garbage collected, but the algorithm only releases memory when no more memory is available. In these environments, it is recommended to always call *Close* when the *ICaptureImage* object is no longer needed, to force the memory to be released immediately.

3.8.2 FileSize

FileSize returns the size of the image file containing the captured image.

Syntax

.Net

C#	`uint FileSize()`
C++	`System::UInt32 FileSize()`
VB	`Function FileSize As UInteger`

ObjC

```
- (uint32_t) fileSize
```

Return Value

The size in bytes of the image file represented by the *ICaptureImage* object.

Remarks

The file size can change when some properties are changed.

3.8.3 SaveToFile

SaveToFile saves the image file to the specified filename path.

Syntax

.Net

C#	`void SaveToFile( string fileName )`
C++	`void SaveToFile( System::String^ fileName )`
VB	`Sub SaveToFile( fileName As String )`

ObjC

```
- (void) saveToFile: (NSString *) fileName
```

Parameters

fileName The filename path of the file to save the image file to.

Remarks

SaveToFile commits any changes to properties to the internal image file before saving the result to the specified filename.

3.8.4 SaveToBuffer

SaveToBuffer saves the image file to the specified memory buffer.

Syntax

.Net

C#	`void SaveToBuffer(System.IntPtr pBuffer, uint size )`
C++	`void SaveToBuffer( System::IntPtr pBuffer, System::UInt32 size )`
VB	`Sub SaveToBuffer( pBuffer As System.IntPtr, size As UInteger )`

ObjC

```
- (void) saveToBuffer: (void *) pBuffer size: (uint32_t) size
```

Parameters

pBuffer A pointer to the memory buffer to copy the image file to.

size	The size of the buffer pointed at by *pBuffer* in bytes. No more than size bytes will be copied to the buffer. If size is larger than the image file size, the extra space will not be touched.

Remarks

SaveToBuffer commits any changes to properties to the internal image file before saving the result to the specified filename. Changes to properties can change the required file size. After changing *ICaptureImage* properties, always call *FileSize* before calling *SaveToBuffer* to get the required file size

3.8.5 GetImageData

GetImageData returns an *IImageData* object that gives access to the data of the captured image.

Syntax

.Net

C#	`IImageData GetImageData()`
C++	`IImageData^ GetImageData()`
VB	`Function GetImageData As IImageData`

ObjC

```
- (P1CaptureCore_CaptureImageData *) getImageData
```

Return Value

An *IImageData* object giving access to the image data of the captured image. A NULL reference is returned if this method is not supported.

3.8.6 GetThumbnail

GetThumbnail returns an *ICaptureImageThumbnail* object representing a thumbnail (reduced resolution) image of the captured image. The width, height and initial color type of the thumbnail are chosen by *ICaptureImage*, but preferred values can also be specified by the caller, which will be used if possible.

Syntax

.Net

C#	`ICaptureImageThumbnail GetThumbnail(`
	`  uint preferredWidth, uint preferredHeight,`
	`  EnumColorType preferredColorType )`

	`ICaptureImageThumbnail GetThumbnail()`

C++	`ICaptureImageThumbnail^ GetThumbnail(`
	`  System::UInt32 preferredWidth, System::UInt32 preferredHeight,`
	`  EnumColorType preferredColorType )`

	`ICaptureImageThumbnail^ GetThumbnail()`

VB	`Function GetThumbnail(`
	`  preferredWidth As UInteger, preferredHeight As UInteger,`
	`  preferredColorType As EnumColorType ) As ICaptureImageThumbnail`

	`Function GetThumbnail As ICaptureImageThumbnail`

ObjC

```
- (P1CaptureCore_CaptureImageThumbnail) getThumbnail
- (P1CaptureCore_CaptureImageThumbnail) getThumbnail:
      (uint32_t) preferredWidth preferredHeight: (uint32_t) preferredHeight
```

```
- (P1CaptureCore_CaptureImageThumbnail) getThumbnail:
      (uint32_t) preferredWidth preferredHeight: (uint32_t) preferredHeight
      preferredColorType: (EnumColorType) preferredColorType
```

Parameters

preferredWidth	The preferred width in pixels of the *ICaptureImageThumbnail* object to return. *ICaptureImage* can choose to ignore this parameter. If zero, or when calling the version of *GetThumbnail* without parameters, *ICaptureImage* will use a default width, which is usually the width of the current embedded thumbnail if any.
preferredHeight	The preferred height in pixels of the *ICaptureImageThumbnail* object to return. *ICaptureImage* can choose to ignore this parameter. If zero, or when calling the version of *GetThumbnail* without any parameters, *ICaptureImage* will use a default height, which is usually the height of the current embedded thumbnail if any.
preferredColorType	The preferred color type of the *ICaptureImageThumbnail* object to return. *ICaptureImage* can choose to ignore this parameter. If *kColorType_Undefined*, or when calling the version of *GetThumbnail* without any parameters, *ICaptureImage* will use a default color type, which is usually the color type of the current embedded thumbnail if any. Note that the color type can be changed later by calling the *ICaptureImageThumbnail* method *SetColorType*.

Return Value

An *ICaptureImageThumbnail* object containing a thumbnail image (reduced resolution) of the captured image. A NULL reference is returned if no thumbnail is available or if one cannot be generated.

3.9 IImageData (P1CaptureCore_ImageData)

The *IImageData* class represents the pixel data of an image or thumbnail. It provides methods for copying the image's pixels and getting and setting properties such as width, height, color, and padding. *IImageData* objects are created and returned by the *ICaptureImage* methods *GetImageData* and *GetThumbnail*.

Members

`ImageType`	Returns the type of the image.
`ColorType (Get/Set)`	Gets or sets the current color type for the image.
`IsColorTypeSupported`	Returns true if a specified color type is supported by the image.
`Width`	Returns the width of the image in pixels.
`Height`	Returns the height of the image in pixels.
`PixelCount`	Returns the number of pixels in the image. Equivalent to *Width* × *Height*.
`Orientation`	Returns the orientation of the image.
`ImageSize`	Returns the total size in bytes of the image, with or without padding.
`LineSize`	Returns the size in bytes of a line in the image, with or without padding.
`PixelSize`	Returns the size of a pixel in the image, with or without padding.
`LinePadding (Get/Set)`	Gets or sets the current amount of padding in bytes to append to each line of the image.
`PixelPadding (Get/Set)`	Gets or sets the current amount of padding in bytes to append to each pixel of the image.
`CopyPixels`	Copies the image pixels to a specified memory buffer, using the current color type, line padding, and pixel padding.
`ToBitmap`	Returns a copy of the image as a .Net *Bitmap* object. (.Net only)
`toNSImage`	Returns a copy of the image as an ObjC *NSImage* object. (ObjC only)

3.9.1 ImageType

ImageType returns the type of the image.

Syntax

.Net

C#	`EnumImageType ImageType()`
C++	`EnumImageType ImageType()`
VB	`Function ImageType As EnumImageType`

ObjC

```
- (EnumColorType) imageType
```

Return Value

An *EnumImageType* indicating the type of the image.

3.9.2 ColorType (Get/Set)

ColorType gets or sets the current color type for the image. The image's color type is used by *CopyPixels* when copying the image, and other methods, such as *PixelSize*, when reporting the size of the image. The color type can often be changed after the image's creation. Supported color types can be determined by using *IsColorTypeSupported*.

Syntax

.Net

C#	`EnumColorType GetColorType()`
	`bool SetColorType( EnumColorType colorType )`
C++	`EnumColorType GetColorType()`
	`bool SetColorType( EnumColorType colorType )`
VB	`Function GetColorType As EnumColorType`
	`Function SetColorType( colorType As EnumColorType ) As Boolean`

ObjC

`- (EnumColorType) colorType`
`- (void) setColorType: (EnumColorType) colorType`

Parameters

colorType The new color type (*EnumColorType*) for the image.

Return Value

GetColorType returns the current color type for the image. *SetColorType* returns true if the *colorType* parameter is supported by the image, otherwise it returns false and does not change the color type.

Remarks

Generally, a image's color type can only be changed to a compatible color type to the original color type, such as from RGB to BGR or from RGB to RGBA. Additionally, the internal representation of the image is usually not altered when setting the color type. The image simply remembers which color type is desired, and returns that color type when calling *CopyPixels*, or uses that color types pixel size when calculating sizes.

3.9.3 IsColorTypeSupported

IsColorTypeSupported returns true if the specified color type is supported by the image.

Syntax

.Net

C#	`bool IsColorTypeSupported( EnumColorType colorType )`
C++	`bool IsColorTypeSupported( EnumColorType colorType )`
VB	`Function IsColorTypeSupported( colorType As EnumColorType ) As Boolean`

ObjC

`- (BOOL) isColorTypeSupported: (EnumColorType) colorType`

Parameters

colorType The color type (*EnumColorType*) to test support for.

Return Value

Returns true if the specified color type is supported by the image, otherwise it returns false.

3.9.4 Width

Width returns the width of the image in pixels.

Syntax

.Net

C#	`uint Width { get; }`
C++	`property System::UInt32 Width` `{ System::UInt32 get(); }`
VB	`ReadOnly Property Width As UInteger`

ObjC

```
- (uint32_t) width
```

Return Value

The width of the image in pixels.

Remarks

The width cannot be changed after the image's creation.

3.9.5 Height

Height returns the height of the image in pixels.

Syntax

.Net

C#	`uint Height { get; }`
C++	`property System::UInt32 Height` `{ System::UInt32 get(); }`
VB	`ReadOnly Property Height As UInteger`

ObjC

```
- (uint32_t) height
```

Return Value

The height of the image in pixels.

Remarks

The height cannot be changed after the image's creation.

3.9.6 PixelCount

PixelCount returns the number of pixels in the image.

Syntax

.Net

C#	`uint PixelCount { get; }`
C++	`property System::UInt32 PixelCount` `{ System::UInt32 get(); }`
VB	`ReadOnly Property PixelCount As UInteger`

ObjC

```
- (uint32_t) pixelCount
```

Return Value

The number of pixels in the image. Equivalent to *Width* × *Height*.

3.9.7 Orientation

Orientation returns the orientation of the image.

Syntax

.Net

C#	`EnumImageOrientation Orientation()`
C++	`EnumImageOrientation Orientation()`
VB	`Function Orientation As EnumImageOrientation`

ObjC

```
- (EnumImageOrientation) orientation
```

Return Value

An *EnumImageOrientation* indicating the orientation of the image. Image orientation is the orientation of the device when the image was captured.

Remarks

To display the image in the correct orientation the image should be rotated by the amount indicated by the *EnumImageOrientation* value.

3.9.8 ImageSize

ImageSize returns the total size in bytes of the image, with or without padding.

Syntax

.Net

C#	`uint ImageSize( bool bIncludePadding )`
C++	`System::UInt32 ImageSize( bool bIncludePadding )`
VB	`Function ImageSize( bIncludePadding As Boolean ) As UInteger`

ObjC

```
- (uint32_t) imageSize: (BOOL) bIncludePadding
```

Parameters

bIncludePadding	If true the return value includes both pixel and line padding, otherwise no padding is included.

Return Value

The total size in bytes of the image. If the *bIncludePadding* parameter is true, the value includes both pixel and line padding, otherwise no padding is included. The current color type (see *ColorType*) is also used for determining the total size.

3.9.9 LineSize

LineSize returns the size in bytes of a line in the image, with or without padding.

Syntax

.Net

C#	`uint LineSize( bool bIncludePadding )`
C++	`System::UInt32 LineSize( bool bIncludePadding )`
VB	`Function LineSize( bIncludePadding As Boolean ) As UInteger`

ObjC

```
- (uint32_t) lineSize: (BOOL) bIncludePadding
```

Parameters

bIncludePadding — If true the return value includes both pixel and line padding, otherwise no padding is included.

Return Value

The size in bytes of a line in the image. If the *bIncludePadding* parameter is true, the value includes both pixel and line padding, otherwise no padding is included. The current color type (see *ColorType*) is also used for determining the line size.

Remarks

A line contains *Width* number of pixels, plus pixel padding and line padding.

3.9.10 PixelSize

PixelSize returns the size of a pixel in the image, with or without padding.

Syntax

.Net

C#	`uint PixelSize( bool bIncludePadding )`
C++	`System::UInt32 PixelSize( bool bIncludePadding )`
VB	`Function PixelSize( bIncludePadding As Boolean ) As UInteger`

ObjC

```
- (uint32_t) pixelSize: (BOOL) bIncludePadding
```

Parameters

bIncludePadding — If true the return value includes pixel padding, otherwise no padding is included.

Return Value

The size of a pixel in the image. If the *bIncludePadding* parameter is true, the value includes pixel padding, otherwise no padding is included. The current color type (see *ColorType*) is also used for determining the pixel size.

3.9.11 LinePadding (Get/Set)

LinePadding gets or sets the current amount of padding in bytes to append to each line of the image.

Syntax

.Net

C#	`uint GetLinePadding()`
	`void SetLinePadding( uint padding )`
C++	`System::UInt32 GetLinePadding()`

	`void SetLinePadding( System::UInt32 padding )`
VB	`Function GetLinePadding As UInteger`
	`Sub SetLinePadding( padding As UInteger )`

ObjC

`- (uint32_t) linePadding`
`- (void) setLinePadding: (uint32_t) padding`

Parameters

padding The amount of line padding in bytes to append to the end of each line in the image.

Return Value

GetLinePadding returns the current line padding in bytes.

3.9.12 PixelPadding (Get/Set)

PixelPadding gets or sets the current amount of padding in bytes to append to each pixel of the image.

Syntax

.Net

C#	`uint GetPixelPadding()`
	`void SetPixelPadding( uint padding )`
C++	`System::UInt32 GetPixelPadding()`
	`void SetPixelPadding( System::UInt32 padding )`
VB	`Function GetPixelPadding As UInteger`
	`Sub SetPixelPadding( padding As UInteger )`

ObjC

`- (uint32_t) pixelPadding`
`- (void) setPixelPadding: (uint32_t)padding`

Parameters

padding The amount of pixel padding in bytes to append to each pixel in the image.

Return Value

GetPixelPadding returns the current pixel padding in bytes.

3.9.13 CopyPixels

CopyPixels copies the image pixels to the specified memory buffer, using the current color type, line padding, and pixel padding.

Syntax

.Net

C#	`void CopyPixels( System.IntPtr pBuffer, uint size )`
C++	`void CopyPixels( System::IntPtr pBuffer, System::UInt32 size )`
VB	`Sub CopyPixels( pBuffer As System.IntPtr, size As UInteger )`

ObjC

`- (void) copyPixels: (void *) pBuffer size: (uint32_t) size`

Parameters

pBuffer	A pointer to the memory buffer to copy the image pixels to.
size	The size of the buffer pointed at by *pBuffer* in bytes. No more than size bytes will be copied to the buffer. If size is larger than the image size including padding, the extra space will not be touched.

Remarks

CopyPixels copies the pixels using the current color type, line padding and pixel padding settings. Remember to set these to their desired values before calling *CopyPixels*. The *size* parameter should be as large as the value returned by *ImageSize* including padding, if the entire image is to be copied.

3.9.14 ToBitmap [.Net Only]

ToBitmap returns a copy of the image as a .Net *Bitmap* object.

Syntax

.Net

C#	`System.Drawing.Bitmap ToBitmap()`
C++	`System::Drawing::Bitmap^ ToBitmap()`
VB	`Function ToBitmap() As System.Drawing.Bitmap`

Return Value

A new *System.Drawing.Bitmap* object with a copy of the image. The *System.Drawing.Bitmap* object has the current color type if it is compatible with *System.Drawing.Bitmap*, otherwise it is converted to the closest compatible color type.

3.9.15 toNSImage [ObjC Only]

toNSImage returns a copy of the image as an ObjC *NSImage* object.

Syntax

ObjC

```
- (NSImage *) toNSImage
```

Return Value

A new *NSImage* object with a copy of the image. The *NSImage* object has the current color type if it is compatible with *NSImage*, otherwise it is converted to the closest compatible color type.

3.10 ICaptureImageThumbnail (P1CaptureCore_CaptureImageThumbnail)

The *ICaptureImageThumbnail* class represents a thumbnail/preview image of a captured image. It contains the thumbnail and provides methods for copying its pixels and getting and setting properties such as width, height, color, and padding. *ICaptureImageThumbnail* objects are created and returned by the *ICaptureImage* method *GetThumbnail*.

ICaptureImageThumbnail inherits from *IImageData*. It currently has no methods of its own. See the documentation for *IImageData* for *ICaptureImageThumbnail* functionality.

Members

Inherited from IImageData	
`ImageType`	Returns the type of the image.
`ColorType (Get/Set)`	Gets or sets the current color type for the image.
`IsColorTypeSupported`	Returns true if a specified color type is supported by the image.
`Width`	Returns the width of the image in pixels.
`Height`	Returns the height of the image in pixels.
`PixelCount`	Returns the number of pixels in the image. Equivalent to *Width* × *Height*.
`Orientation`	Returns the orientation of the image.
`ImageSize`	Returns the total size in bytes of the image, with or without padding.
`LineSize`	Returns the size in bytes of a line in the image, with or without padding.
`PixelSize`	Returns the size of a pixel in the image, with or without padding.
`LinePadding (Get/Set)`	Gets or sets the current amount of padding in bytes to append to each line of the image.
`PixelPadding (Get/Set)`	Gets or sets the current amount of padding in bytes to append to each pixel of the image.
`CopyPixels`	Copies the image pixels to a specified memory buffer, using the current color type, line padding, and pixel padding.
`ToBitmap`	Returns a copy of the image as a .Net *Bitmap* object. (.Net only)
`toNSImage`	Returns a copy of the image as an ObjC *NSImage* object. (ObjC only)

3.11 ICaptureObject (P1CaptureCore_CaptureObject)

The *ICaptureObject* base class provides a common set of functionality that is shared between the main capture objects of CaptureCore: *ICaptureProvider*, *ICamera* and *ICaptureImage*. It does not exist as an object on its own, and is only accessible via a derived class.

ICaptureObject inherits from *IErrorSource* and *IEventSource*. It is a parent to *ICapability* (*ICapabilityList*) and *IProperty* (*IPropertyList*) objects.

Members

`Id`	Returns an unique ID representing the *ICaptureObject* instance.
`GetCapabilityList`	Returns a reference to a *ICapabilityList* object containing all *ICapability* objects for this *ICaptureObject* object.
`GetPropertyList`	Returns a reference to a *IPropertyList* object containing all *IProperty* objects for this *ICaptureObject* object.
`GetCapability`	Returns a reference to an *ICapability* object for this *ICaptureObject* object, with a specified ID.
`GetProperty`	Returns a reference to an *IProperty* object for this *ICaptureObject* object, with a specified ID.
Inherited from IErrorSource	
`GetError`	Returns the next *IErrorObject* object, if any, for the *ICaptureObject* object.
Inherited from IEventSource	
`AddReceiver`	Attaches an *IEventReceiver* object to receive events (*IEventObject*) from the *ICaptureObject* object.
`RemoveReceiver`	Detaches a previously attached *IEventReceiver* object so that it no longer receives events (*IEventObject*) from the *ICaptureObject* object.

Events

General `(EnumCaptureObjectEventId)`	
`kCaptureObjectEvent_CapabilityChange`	A capability's value has changed.
`kCaptureObjectEvent_PropertyChange`	A property's value has changed.
`kCaptureObjectEvent_SettingDescriptorChange`	A property's setting descriptor has changed.

3.11.1 Id

Id returns an unique ID representing the *ICaptureObject* instance.

Syntax

.Net

C#	`uint Id { get; }`
C++	`property System::UInt32 Id` `{ System::UInt32 get(); }`
VB	`ReadOnly Property Id As UInteger`

ObjC

```
- (uint32_t) id
```

Return Value

A number that uniquely identifies the *ICaptureObject* instance.

Remarks

No two instances of *ICaptureObject* classes will have the same ID and it doesn't change after the object is created.

3.11.2 GetCapabilityList

GetCapabilityList returns a reference to a *ICapabilityList* object containing all *ICapability* objects for this *ICaptureObject* object.

Syntax

.Net

C#	`ICapabilityList  GetCapabilityList()`
C++	`ICapabilityList^ GetCapabilityList()`
VB	`Function GetCapabilityList As ICapabilityList`

ObjC

```
- (P1CaptureCore_CapabilityList *) getCapabilityList
```

Return Value

An *ICapabilityList* object containing all the *ICapability* objects for the object. A NULL reference or an empty list is returned if there are no defined capabilities for the object.

Remarks

The returned *ICapabilityList* will be a copy of an internal capability list, if the number of capabilities of the *ICaptureObject* object can change dynamically. This allows the returned list to be used without any problems that may arise from dynamic changes – only the internal list is changed dynamically. However, this is very unusual and generally the capability list will be the same on each call to *GetCapabilityList*. Nevertheless, applications should avoid caching the returned *ICapabilityList* object, and retrieve a new list when it is needed.

3.11.3 GetPropertyList

GetPropertyList returns a reference to a *IPropertyList* object containing all *IProperty* objects for this *ICaptureObject* object.

Syntax

.Net

C#	`IPropertyList  GetPropertyList()`
C++	`IPropertyList^ GetPropertyList()`
VB	`Function GetPropertyList As IPropertyList`

ObjC

```
- (P1CaptureCore_PropertyList *) getPropertyList
```

Return Value

An *IPropertyList* object containing all the *IProperty* objects for the object. A NULL reference or an empty list is returned if there are no defined properties for the object.

Remarks

The returned *IPropertyList* will be a copy of an internal property list, if the number of properties of the *ICaptureObject* object can change dynamically. This allows the returned list to be used without any problems that may arise from dynamic changes – only the internal list is changed dynamically. Applications should avoid caching the returned *IPropertyList* object, and retrieve a new list when it is needed.

3.11.4 GetCapability

GetCapability returns a reference to an *ICapability* object for this *ICaptureObject* object, with a specified ID.

Syntax

.Net

C#	`ICapability  GetCapability( uint capabilityID )`
C++	`ICapability^ GetCapability( System::UInt32 capabilityID )`
VB	`Function GetCapability( capabilityID As UInteger ) As ICapability`

ObjC

```
- (P1CaptureCore_Capability *) getCapability: (uint32_t) capabilityID
```

Parameters

capabilityID — The capability ID of the *ICapability* object to return. Capability IDs are specific for each class derived from *ICaptureObject*. See Capability Reference for possible capability IDs.

Return Value

An *ICapability* object with the specified capability ID, if it is present in the capability list for the object. If the specified capability is not present, then a NULL reference is returned.

Remarks

GetCapability is equivalent to calling *GetCapabilityList*, and calling the *ICapabilityList* method *GetCapabiltity*, which iterates through the capability list searching for the specified capability ID.

3.11.5 GetProperty

GetProperty returns a reference to an *IProperty* object for this *ICaptureObject* object, with a specified ID.

Syntax

.Net

C#	`IProperty  GetProperty( uint propertyID )`
C++	`IProperty^ GetProperty( System::UInt32 propertyID )`
VB	`Function GetProperty( propertyID As UInteger ) As IProperty`

ObjC

```
- (P1CaptureCore_Property *) getProperty: (uint32_t) propertyID
```

Parameters

propertyID — The property ID of the *IProperty* object to return. Property IDs are specific for each class derived from *ICaptureObject*. See Property Reference for possible property IDs.

Return Value

An *IProperty* object with the specified property ID, if it is present in the property list for the object. If the specified property is not present, then a NULL reference is returned.

Remarks

GetProperty is equivalent to calling *GetPropertyList*, and calling the *IPropertyList* method *GetProperty*, which iterates through the property list searching for the specified property ID.

3.11.6 kCaptureObjectEvent_CapabilityChange

This event is posted by the *ICaptureObject* object when an *ICapability* object, owned by the *ICaptureObject* object, has changed in some way.

Arguments

0 The capability ID of the *ICapability* object that has changed.

3.11.7 kCaptureObjectEvent_PropertyChange

This event is posted by the *ICaptureObject* object when an *IProperty* object, owned by the *ICaptureObject* object, has changed in some way.

Arguments

0 The property ID of the *IProperty* object has changed.

3.11.8 kCaptureObjectEvent_SettingDescriptorChange

This event is posted by the *ICaptureObject* object when the *ISettingDescriptor* object of an *IProperty* object, owned by the *ICaptureObject*, has changed in some way.

Arguments

0 The property ID of the *IProperty* object that owns the *ISettingDescriptor* that has changed.

3.12 ICapabilityList (P1CaptureCore_CapabilityList)

The *ICapabilityList* class is a list container for *ICapability* objects. It is a child object of *ICaptureObject*, and inherits from *IChildObject* and *IObjectList*.

Members

`GetCapability`	Returns a reference to an *ICapability* object in the list with the specified ID.
`Dump`	Dumps debug information for all capabilities in the list to a debug monitor or to a log file specified through the *ICaptureCore* object.
Inherited from IChildObject	
`Parent`	Returns the parent *ICaptureObject* object of this object.
Inherited from IObjectList	
`Size`	Returns the number of *ICapability* items in the list.
`IsEmpty`	Returns true if the list is empty.
`First`	Returns a reference to the first *ICapability* item in the list.
`Last`	Returns a reference to the last *ICapability* item in the list.
`Next`	Returns a reference to the next *ICapability* item in the list following a specified *ICapability* item already in the list.
`Previous`	Returns a reference to the previous *ICapability* item in the list preceding a specified *ICapability* item already in the list.
`Insert`	Inserts a new item in front of another specified *ICapability* item in the list. Requires insert access rights.
`Remove`	Removes a specified *ICapability* item from the list. Requires remove access rights.
`Clear`	Removes all items from the list. Requires remove access rights.
`GetAccess`	Returns the access rights for this list as a bitmask of *EnumListAccess* values.
`HasAccess`	Returns true if the list allows the specified access rights.

3.12.1 GetCapability

GetCapability returns a reference to an *ICapability* object in the list with the specified ID.

Syntax

.Net

C#	`ICapability  GetCapability( uint capabilityID )`
C++	`ICapability^ GetCapability( System::UInt32 capabilityID )`
VB	`Function GetCapability( capabilityID As UInteger ) As ICapability`

ObjC

```
- (P1CaptureCore_Capability *) getCapability: (uint32_t) capabilityID
```

Parameters

capabilityID The capability ID of the *ICapability* object to return.

Return Value

An *ICapability* object with the specified capability ID, if it is present in the capability list. If the specified capability is not present, then a NULL reference is returned.

3.12.2 Dump

Dump dumps debug information for all capabilities in the list to a debug monitor and to a log file specified through the *ICaptureCore* object.

Syntax

.Net

C#	`void Dump()`
C++	`void Dump()`
VB	`Sub Dump`

ObjC

```
- (void) dump
```

Remarks

Dump calls the *ICapability* method *Dump* on each *ICapability* object in the list. This outputs a textual description of each capability in the list to the platform's debug monitor, and to a log file, if one has been setup via the *LogMsgFileName* method of *ICaptureCore*.

3.13 ICapability (P1CaptureCore_Capability)

Objects of the *ICapability* class each represent a single capability of their parent *ICaptureObject*. A capability is a read-only value that tells something about what an instance of an *ICaptureObject* is capable of doing. Most capabilities are Boolean values, but they may also be numbers, strings and other value types. Each capability has an unique ID; see the Capability Reference section for possible capabilities for each *ICaptureObject* derived class.

Capabilities are different than properties (see *IProperty*). Capabilities tell what an object can do, whereas properties provide settings and information about the object. Capabilities are normally used for determining if certain methods can be called, or enabling/disabling some functionality. They are generally only used to control the logic of an application, and are not directly presented to a user. Properties are often settings or information about an object, and are often presented to a user, in addition to possibly controlling the logic of an application.

Capabilities can change dynamically in response to method calls or events. A capability change event (*kCaptureObjectEvent_CapabilityChange*) is posted by the parent *ICaptureObject* whenever a capability changes.

ICapability is a child object of *ICaptureObject*, and inherits from *IChildObject* and *IValueRead*.

Members

`Id`	Returns the unique capability ID for this capability.
`Name`	Returns a string name for the capability.
`Unit`	Returns a string containing an optional unit for the capability.
`Dump`	Dumps debug information for the capability to a debug monitor and to a log file specified through the *ICaptureCore* object.
Inherited from IChildObject	
`Parent`	Returns the parent *ICaptureObject* object of this object.
Inherited from IValueRead	
`ValueType`	Returns the value type (Boolean, integer, string, etc) of the object.
`IsUndefined`	Returns true if the object's value is undefined.
`GetValue`	Gets the value of the object if the type of the object and the type passed to *GetValue* are compatible. One can always get a string representation for all value types. *GetValue* is available on platforms that support overloading.
`GetValueBool`	Returns the value of the object if its value type is a Boolean.
`GetValueInt32`	Returns the value of the object if its value type is a 32-bit signed integer (or enumeration).
`GetValueUInt32`	Returns the value of the object if its value type is a 32-bit unsigned integer (or enumeration).
`GetValueInt64`	Returns the value of the object if its value type is a 64-bit signed integer.

GetValueUInt64	Returns the value of the object if its value type is a 64-bit unsigned integer.
GetValueFloat64	Returns the value of the object if its value type is a 64-bit floating point.
GetValueString	Returns the value of the object if its value type is a string, or a string representation of the value for all other value types.
GetValueEnum	Returns the value of the object if its value type is an enumeration (32-bit signed integer).
GetValuePoint	Returns the value of the object if its value type is a point (32-bit signed integer).
GetValuePointFloat	Returns the value of the object if its value type is a point (64-bit floating point).
GetValueArea	Returns the value of the object if its value type is an area (32-bit signed integer).
GetValueAreaFloat	Returns the value of the object if its value type is an area (64-bit floating point).
GetValueRect	Returns the value of the object if its value type is a rectangle (32-bit signed integer).
GetValueRectFloat	Returns the value of the object if its value type is a rectangle (64-bit floating point).
Compare	Compares this object's value to another object of the same value type, returning a signed integer representing if this object is less than, greater than or equal to the other object.

3.13.1 Id

Id returns the unique capability ID for this *ICapability* object.

Syntax

.Net

C#	`uint Id { get; }`
C++	`property System::UInt32 Id { System::UInt32 get(); }`
VB	`ReadOnly Property Id As UInteger`

ObjC

```
- (uint32_t) id
```

Return Value

A number representing the unique capability ID of the object. See Capability Reference for further details about capability IDs for different classes.

Remarks

The capability ID for an *ICapability* object doesn't change after the object is created.

3.13.2 Name

Name returns a display name string for the capability. The default name is vendor specific, but alternative names can also be returned.

Syntax

.Net

C#	`string Name { get; }`
	`string NameEx( EnumCaptureCoreName which )`
C++	`property System::String^ Name` `{ System::String^ get(); }`
	`System::String^ NameEx( EnumCaptureCoreName which )`
VB	`ReadOnly Property Name As String`
	`Function NameEx( EnumCaptureCoreName which ) As String`

ObjC

`- (NSString *) name`
`- (NSString *) name: (EnumCaptureCoreName) which`

Parameters

which	An *EnumCaptureCoreName* value specifying which name to return. If the *which* parameter is not specified, the returned name is vendor specific (*CaptureCoreName_VendorSpecific*).

Return Value

A string containing a display name for the capability. The specific name returned is specified by the optional parameter *which*. The default name is vendor specific.

Remarks

[.Net only] The method that takes the parameter *which* is called *NameEx*, to avoid conflicting with the *Name* property.

There are several possible names for a capability: vendor specific, long or short. Vendor specific names are defined by the manufacturer associated with the object. Long and short names are defined by Phase One and are generally common for all objects. Short names are guaranteed to be 20 characters or less. The names for each name type may be the same.

The names for an *ICapability* object do not change after the object is created.

3.13.3 Unit

Unit returns a string containing an optional unit for the capability.

Syntax

.Net

C#	`string Unit { get; }`
C++	`property System::String^ Unit` `{ System::String^ get(); }`
VB	`ReadOnly Property Unit As String`

ObjC

`- (NSString *) unit`

Return Value

A string containing an optional display unit for the capability, such as degrees, Celsius, pixels, and so on. The string can be empty if no unit is defined.

Remarks

The unit for an *ICapability* object doesn't change after the object is created.

3.13.4 Dump

Dump dumps debug information for the capability to a debug monitor and to a log file specified through the *ICaptureCore* object.

Syntax

.Net

C#	`void Dump()`
C++	`void Dump()`
VB	`Sub Dump`

ObjC

```
- (void) dump
```

Remarks

Dump outputs a text message describing the Name and value of the *ICapability* object. The message is output to the platform's debug monitor, and written to a log file, if one has been setup via the *LogMsgFileName* method of *ICaptureCore*.

3.14 IPropertyList (P1CaptureCore_PropertyList)

The *IPropertyList* class is a list container for *IProperty* objects. It is a child object of *ICaptureObject*, and inherits from *IChildObject* and *IObjectList*.

Members

`GetProperty`	Returns a reference to an *IProperty* object in the list with the specified ID.
`RestoreDefault`	Resets all properties in the list to their default value, if they have a default value.
`Refresh`	Reloads all property values from their data source. *Refresh* is not necessary for retrieving a property's value. This method can affect performance, so it should only be called when a user specifically requests a refresh or manual synchronization.
`Dump`	Dumps debug information for all properties in the list to a debug monitor or to a log file specified through the *ICaptureCore* object.
Inherited from IChildObject	
`Parent`	Returns the parent *ICaptureObject* object of this object.
Inherited from IObjectList	
`Size`	Returns the number of *IProperty* items in the list.
`IsEmpty`	Returns true if the list is empty.
`First`	Returns a reference to the first *IProperty* item in the list.
`Last`	Returns a reference to the last *IProperty* item in the list.
`Next`	Returns a reference to the next *IProperty* item in the list following a specified *IProperty* item already in the list.
`Previous`	Returns a reference to the previous *IProperty* item in the list preceding a specified *IProperty* item already in the list.
`Insert`	Inserts a new item in front of another specified *IProperty* item in the list. Requires insert access rights.
`Remove`	Removes a specified *IProperty* item from the list. Requires remove access rights.
`Clear`	Removes all items from the list. Requires remove access rights.
`GetAccess`	Returns the access rights for this list as a bitmask of *EnumListAccess* values.
`HasAccess`	Returns true if the list allows the specified access rights.

3.14.1 GetProperty

GetProperty returns a reference to an *IProperty* object in the list with the specified ID.

Syntax

.Net

C#	`IProperty  GetProperty( uint propertyID )`

C++	`IProperty^ GetProperty( System::UInt32 propertyID )`
VB	`Function GetProperty( propertyID As UInteger ) As IProperty`

ObjC

```
- (P1CaptureCore_Property *) getProperty: (uint32_t) propertyID
```

Parameters

propertyID The property ID of the *IProperty* object to return.

Return Value

An *IProperty* object with the specified property ID, if it is present in the property list. If the specified property is not present, then a NULL reference is returned.

3.14.2 RestoreDefault

RestoreDefault resets all properties in the list to their default value, if they have a default value.

Syntax

.Net

C#	`void RestoreDefault()`
C++	`void RestoreDefault()`
VB	`Sub RestoreDefault`

ObjC

```
- (void) restoreDefault
```

Remarks

RestoreDefault calls the *IProperty* method *RestoreDefault* on each *IProperty* object in the list.

3.14.3 Refresh

Refresh reloads all property values from their data source. *Refresh* is not necessary for retrieving a property's value. This method can affect performance, so it should only be called when a user specifically requests a refresh or manual synchronization.

Normally, properties are automatically synchronized with their source. However, this is not always the case. In the event that properties fall out of synchronization with the source/device for the property, this method can be used to reload their values from the source/device.

Syntax

.Net

C#	`void Refresh()`
C++	`void Refresh()`
VB	`Sub Refresh`

ObjC

```
- (void) refresh
```

Remarks

Refresh calls the *IProperty* method *Refresh* on each *IProperty* object in the list.

A call to *Refresh* may take some time, and is only necessary if property objects are out of synchronization with their data source. Since this is unusual, it is recommended to only call this method, if the user requests synchronization manually.

3.14.4 Dump

Dump dumps debug information for all properties in the list to a debug monitor or to a log file specified through the *ICaptureCore* object.

Syntax

.Net

C#	`void Dump()`
C++	`void Dump()`
VB	`Sub Dump`

ObjC

```
- (void) dump
```

Remarks

Dump calls the *IProperty* method *Dump* on each *IProperty* object in the list. This outputs a textual description of each property in the list to the platform's debug monitor, and to a log file, if one has been setup via the *LogMsgFileName* method of *ICaptureCore*.

3.15 IProperty (P1CaptureCore_Property)

Objects of the *IProperty* class each represent a single property of their parent *ICaptureObject*. A property represents a setting or some other information about an instance of an *ICaptureObject*, and may be a number, string, Boolean, or other value type. Each property has an unique ID; see the Property Reference section for possible properties for each *ICaptureObject* derived class.

Properties are different than capabilities (see *ICapability*). Capabilities tell what an object can do, whereas properties provide settings and information about the object, and are often presented to a user, in addition to possibly controlling the logic of an application.

Properties can be read-only or writeable. In addition, they may have an optional setting descriptor (see *ISettingDescriptor*) describing the valid range or values of the property.

Properties can change dynamically in response to method calls or events. Not only can their value change, but also their setting descriptor, and whether they are read-only or writeable. A property or setting descriptor change event (*kCaptureObjectEvent_PropertyChange* or *kCaptureObjectEvent_SettingDescriptorChange*) is posted by the parent *ICaptureObject* whenever the property or setting descriptor for a property changes.

IProperty is a child object of *ICaptureObject*, and inherits from *IChildObject*, *IValueRead* and *IValueWrite*. It is a parent to *ISettingDescriptor* and *ISettingValue* (*ISettingValueList*) objects.

Members

`Id`	Returns the unique property ID for this property.
`Name`	Returns a string name for the property.
`Unit`	Returns a string containing an optional unit for the property.
`GetSetttingDescriptor`	Returns an optional *ISettingDescriptor* object describing the values and/or range that the property can be set to.
`IsDisabled`	Returns true if the property is currently disabled, that is that its value cannot be written to or read from.
`IsDefaultValue`	Returns true if the property's current value is the same as its default value. It always returns false if it does not have a default value.
`RestoreDefault`	Sets the property to its default value, if it has one.
`Refresh`	Reloads the property's value from its data source. *Refresh* is not necessary for retrieving the property value. This method can affect performance, so it should only be called when a user specifically requests a refresh or manual synchronization.
`Dump`	Dumps debug information for the property to a debug monitor and to a log file specified through the *ICaptureCore* object.
Inherited from IChildObject	
`Parent`	Returns the parent *ICaptureObject* object of this object.
Inherited from IValueRead	

`ValueType`	Returns the value type (Boolean, integer, string, etc) of the object.
`IsUndefined`	Returns true if the object's value is undefined.
`GetValue`	Gets the value of the object if the type of the object and the type passed to *GetValue* are compatible. One can always get a string representation for all value types. *GetValue* is available on platforms that support overloading.
`GetValueBool`	Returns the value of the object if its value type is a Boolean.
`GetValueInt32`	Returns the value of the object if its value type is a 32-bit signed integer (or enumeration).
`GetValueUInt32`	Returns the value of the object if its value type is a 32-bit unsigned integer (or enumeration).
`GetValueInt64`	Returns the value of the object if its value type is a 64-bit signed integer.
`GetValueUInt64`	Returns the value of the object if its value type is a 64-bit unsigned integer.
`GetValueFloat64`	Returns the value of the object if its value type is a 64-bit floating point.
`GetValueString`	Returns the value of the object if its value type is a string, or a string representation of the value for all other value types.
`GetValueEnum`	Returns the value of the object if its value type is an enumeration (32-bit signed integer).
`GetValuePoint`	Returns the value of the object if its value type is a point (32-bit signed integer).
`GetValuePointFloat`	Returns the value of the object if its value type is a point (64-bit floating point).
`GetValueArea`	Returns the value of the object if its value type is an area (32-bit signed integer).
`GetValueAreaFloat`	Returns the value of the object if its value type is an area (64-bit floating point).
`GetValueRect`	Returns the value of the object if its value type is a rectangle (32-bit signed integer).
`GetValueRectFloat`	Returns the value of the object if its value type is a rectangle (64-bit floating point).
`Compare`	Compares this object's value to another object of the same value type, returning a signed integer representing if this object is less than, greater than or equal to the other object.
Inherited from IValueWrite	
`IsReadOnly`	Returns true if the object is a read-only object. Set value methods may not be called on a read-only object.

`SetValue`	Sets the value of the object if the type of the object and the type passed to *SetValue* are compatible. *SetValue* is available on platforms that support overloading.
`SetValueBool`	Sets the value of the object if its value type is a Boolean.
`SetValueInt32`	Sets the value of the object if its value type is a 32-bit signed integer (or enumeration).
`SetValueUInt32`	Sets the value of the object if its value type is a 32-bit unsigned integer (or enumeration).
`SetValueInt64`	Sets the value of the object if its value type is a 64-bit signed integer.
`SetValueUInt64`	Sets the value of the object if its value type is a 64-bit unsigned integer.
`SetValueFloat64`	Sets the value of the object if its value type is a 64-bit floating point.
`SetValueString`	Sets the value of the object if its value type is a string.
`SetValueEnum`	Sets the value of the object if its value type is an enumeration (32-bit signed integer).
`SetValuePoint`	Sets the value of the object if its value type is a point (32-bit signed integer).
`SetValuePointFloat`	Sets the value of the object if its value type is a point (64-bit floating point).
`SetValueArea`	Sets the value of the object if its value type is an area (32-bit signed integer).
`SetValueAreaFloat`	Sets the value of the object if its value type is an area (64-bit floating point).
`SetValueRect`	Sets the value of the object if its value type is a rectangle (32-bit signed integer).
`SetValueRectFloat`	Sets the value of the object if its value type is a rectangle (64-bit floating point).

3.15.1 Id

Id returns the unique property ID for this *IProperty* object.

Syntax

.Net

C#	`uint Id { get; }`
C++	`property System::UInt32 Id` `{ System::UInt32 get(); }`
VB	`ReadOnly Property Id As UInteger`

ObjC

```
- (uint32_t) id
```

Return Value

A number representing the unique property ID for this object. See Property Reference for further details about property IDs for different classes.

Remarks

The property ID for an *IProperty* object doesn't change after the object is created.

3.15.2 Name

Name returns a display name string for the property. The default name is vendor specific, but alternative names can also be returned.

Syntax

.Net

C#	`string Name { get; }`
	`string NameEx( EnumCaptureCoreName which )`
C++	`property System::String^ Name` `{ System::String^ get(); }`
	`System::String^ NameEx( EnumCaptureCoreName which )`
VB	`ReadOnly Property Name As String`
	`Function NameEx( EnumCaptureCoreName which ) As String`

ObjC

`- (NSString *) name`
`- (NSString *) name: (EnumCaptureCoreName) which`

Parameters

which	An *EnumCaptureCoreName* value specifying which name to return. If the *which* parameter is not specified, the returned name is vendor specific (*CaptureCoreName_VendorSpecific*).

Return Value

A string containing a display name for the property. The specific name returned is specified by the optional parameter *which*. The default name is vendor specific.

Remarks

[.Net only] The method that takes the parameter *which* is called *NameEx*, to avoid conflicting with the *Name* property.

There are several possible names for a property: vendor specific, long or short. Vendor specific names are defined by the manufacturer associated with the object. Long and short names are defined by Phase One and are generally common for all objects. Short names are guaranteed to be 20 characters or less. The names for each name type may be the same.

The names for an *IProperty* object do not change after the object is created.

3.15.3 Unit

Unit returns a string containing an optional unit for the property.

Syntax

.Net

C#	`string Unit { get; }`

C++	`property System::String^ Unit` `{ System::String^ get(); }`
VB	`ReadOnly Property Unit As String`

ObjC

```
- (NSString *) unit
```

Return Value

A string containing an optional display unit for the property, such as degrees, Celsius, pixels, and so on. The string can be empty if no unit is defined.

Remarks

The unit for an *IProperty* object doesn't change after the object is created.

3.15.4 GetSettingDescriptor

GetSettingDescriptor returns an optional *ISettingDescriptor* object describing the values and/or range that the property can be set to.

Syntax

.Net

C#	`ISettingDescriptor  GetSettingDescriptor()`
C++	`ISettingDescriptor^ GetSettingDescriptor()`
VB	`Function GetSettingDescriptor As ISettingDescriptor`

ObjC

```
- (P1CaptureCore_SettingDescriptor *) getSettingDescriptor
```

Return Value

A reference to a *ISettingDescriptor* object for the *IProperty* object. A NULL reference is returned if no setting descriptor is defined for the *IProperty* object.

Remarks

If *GetSettingDescriptor* returns a NULL reference, then no setting descriptor is defined for the property. If the property is not read-only, this means that it can be set to any legal value for the value type of the property.

3.15.5 IsDisabled

IsDisabled returns true if the property is currently disabled, that is that its value cannot be written to or read from.

Properties can become disabled if the current state of the parent object does not allow the property to be set (or read). A property's disabled state can change at any moment. If it does change a *kCaptureObjectEvent_PropertyChange* event is sent by the parent object.

Syntax

.Net

C#	`bool IsDisabled()`
C++	`bool IsDisabled()`
VB	`Function IsDisabled As Boolean`

ObjC

```
- (BOOL) isDisabled
```

Return Value

True if the property is currently not accessible or not available. The return value can change instantaneously or depend upon the values of other properties.

Remarks

A disabled property is automatically read-only (*IsReadOnly* will return true). However, unlike a read-only property, a disabled property can't generally be read from, and may even throw an exception if its value is read.

3.15.6 IsDefaultValue

IsDefaultValue returns true if the property's current value is the same as its default value. It always returns false if it does not have a default value. A property's default value is defined in its setting descriptor.

Syntax

.Net

C#	`bool IsDefaultValue()`
C++	`bool IsDefaultValue()`
VB	`Function IsDefaultValue As Boolean`

ObjC

```
- (BOOL) isDefaultValue
```

Return Value

True if there is a default value and the property value is the same as the default.

3.15.7 RestoreDefault

RestoreDefault sets the property to its default value, if it has a one. A property's default value is defined in its setting descriptor.

Syntax

.Net

C#	`void RestoreDefault()`
C++	`void RestoreDefault()`
VB	`Sub RestoreDefault`

ObjC

```
- (void) restoreDefault
```

Remarks

RestoreDefault tests if there is an *ISettingDescriptor* for the property, and retrieves the default value by calling the *ISettingDescriptor* method *Default*. If there is not *ISettingDescriptor* or the descriptor does not have a default, then *RestoreDefault* does nothing.

3.15.8 Refresh

Refresh reloads the property's value from its data source. *Refresh* is not necessary for retrieving the property value. This method can affect performance, so it should only be called when a user specifically requests a refresh or manual synchronization.

Normally, properties are automatically synchronized with their source. However, this is not always the case. In the event that a property falls out of synchronization with the source/device for the property, this method can be used to reload its value from the source/device.

Syntax

.Net

C#	`void Refresh()`
C++	`void Refresh()`
VB	`Sub Refresh`

ObjC

```
- (void) refresh
```

Remarks

A call to *Refresh* may take some time, and is only necessary if the property object is out of synchronization with its data source. Since this is unusual, it is recommended to only call this method, if the user requests synchronization manually.

3.15.9 Dump

Dump dumps debug information for the property to a debug monitor and to a log file specified through the *ICaptureCore* object.

Syntax

.Net

C#	`void Dump()`
C++	`void Dump()`
VB	`Sub Dump`

ObjC

```
- (void) dump
```

Remarks

Dump outputs a text message describing the Name and value of the *IProperty* object. The message is output to the platform's debug monitor, and written to a log file, if one has been setup via the *LogMsgFileName* method of *ICaptureCore*.

3.16 ISettingDescriptor (P1CaptureCore_SettingDescriptor)

The *ISettingDescriptor* class provides optional information about the possible values that a specific *IProperty* object can be set to. It can describe an *IProperty* object's valid range and/or provide a list of values, as well as describing a default value.

ISettingDescriptor objects are read-only objects, however their values can change dynamically, in response to method calls or events. A setting descriptor change event (*kCaptureObjectEvent_SettingDescriptorChange*) is sent by the *ICaptureObject* object that is indirectly the parent of the *ISettingDescriptor* object, whenever the setting descriptor changes.

An application can use the methods of *ISettingDescriptor* to determine which UI control could be appropriate for controlling the value of an *IProperty* object. If there is no *ISettingDescriptor* object for an *IProperty* object, or the *ISettingDescriptor* object does not have a value list (see *HasValueList* and *GetValueList*), then an edit control is usually appropriate to display and edit a property's value. If the *ISettingDescriptor* object does have a value list, then there are generally two options. If the *IsValueListSelectOnly* method returns true, indicating that only values from the value list may be selected, then a drop-down list control is useful. If *IsValueListSelectOnly* returns false, a combination edit/drop-down list control will allow a user to both select values and edit values.

IProperty objects can also have a default value and a range limit. If there is no range limit, then all values are considered in the range. An application can test if a value is valid for a specific *IProperty* object by calling the method *ValidateValue*, which throws an error if the value is not a valid setting for the *IProperty* object. Note that an *IProperty* object calls *ValidateValue* automatically whenever the application sets it to a new value.

ISettingDescriptor is a child object of *IProperty*, and inherits from *IChildObject*. It is a parent to *ISettingValue* (*ISettingValueList*) objects.

Members

`ValueType`	Returns the value type (Boolean, number, string, etc) of the *ISettingDescriptor* object. This is always the same as the parent *IProperty* object's value type.
`HasDefault`	Returns true if the parent *IProperty* object has a default value. If true, then *Default* can be used to retrieve the default value.
`Default`	Returns the default value for the parent *IProperty* object, if it has one.
`HasRange`	Returns true if the parent *IProperty* object has a range limit. If true, then *RangeMinimum* and *RangeMaximum* can be used to retrieve the minimum and maximum range values.
`RangeMinimum`	Returns the minimum range value for the parent *IProperty* object, if it has one.
`RangeMaximum`	Returns the maximum range value for the parent *IProperty* object, if it has one.
`HasValueList`	Returns true if there is a list of values that the parent *IProperty* object can be set to. If true, then *GetValueList* can be used to retrieve the list.

`IsValueListSelectOnly`	Returns true if the parent *IProperty* object can only be set to values from the list returned by *GetValueList*.
`GetValueList`	Returns an optional list of values that the parent *IProperty* object can be set to.
`ValidateValue`	Tests whether an *IValueRead* object has a value that is valid for the parent *IProperty* object, and throws an exception if it is not. A value is valid if it is in range or in the value list, if any.
Inherited from IChildObject	
`Parent`	Returns the parent *IProperty* object of this object.

3.16.1 ValueType

ValueType returns the value type (Boolean, number, string, etc) of the *ISettingDescriptor* object. This is always the same as the parent *IProperty* object's value type.

Syntax

.Net

C#	`EnumValueType ValueType { get; }`
C++	`property EnumValueType^ ValueType` `{ EnumValueType^ get(); }`
VB	`ReadOnly Property ValueType As EnumValueType`

ObjC

```
- (EnumValueType) valueType
```

Return Value

An *EnumValueType* enumeration value that indicates which value type the *ISettingDescriptor* and its parent *IProperty* object are.

Remarks

All *ISettingValue* child objects of an *ISettingDescriptor* object have the same value type as the *ISettingDescriptor* object.

The value type of an *ISettingDescriptor* object doesn't change after the object is created.

3.16.2 HasDefault

HasDefault returns true if the parent *IProperty* object has a default value. If true, then *Default* can be used to retrieve the default value.

Syntax

.Net

C#	`bool HasDefault()`
C++	`bool HasDefault()`
VB	`Function HasDefault As Boolean`

ObjC

```
- (BOOL) hasDefault
```

Return Value

True if the *ISettingDescriptor* object contains a default value for the parent *IProperty* object.

3.16.3 Default

Default returns the default value for the parent *IProperty* object, if it has one.

Syntax

.Net

C#	`ISettingValue  Default()`
C++	`ISettingValue^ Default()`
VB	`Function Default As ISettingValue`

ObjC

```
- (P1CaptureCore_SettingValue *) default
```

Return Value

A reference to an *ISettingValue* object that represents the default value for the parent *IProperty* object. A NULL reference is returned if the property doesn't have a default value.

Remarks

An application can call *HasDefault* to test if there is a default value.

3.16.4 HasRange

HasRange returns true if the parent *IProperty* object has a range limit. If true, then *RangeMinimum* and *RangeMaximum* can be used to retrieve the minimum and maximum range values.

Syntax

.Net

C#	`bool HasRange()`
C++	`bool HasRange()`
VB	`Function HasRange As Boolean`

ObjC

```
- (BOOL) hasRange
```

Return Value

True if the *ISettingDescriptor* object has a range limit, that is it contains a minimum and maximum range value for the parent *IProperty* object.

3.16.5 RangeMinimum

RangeMinimum returns the minimum range value for the parent *IProperty* object, if it has one.

Syntax

.Net

C#	`ISettingValue  RangeMinimum()`
C++	`ISettingValue^ RangeMinimum()`
VB	`Function RangeMinimum As ISettingValue`

ObjC

```
- (P1CaptureCore_SettingValue *) rangeMinimum
```

Return Value

A reference to an *ISettingValue* object that represents the minimum value for the parent *IProperty* object. A NULL reference is returned if the property doesn't have a range limit.

Remarks

If there is a range limit, then both the minimum and maximum range values will be defined. An application can call *HasRange* to test if there is a range limit.

3.16.6 RangeMaximum

RangeMaximum returns the maximum range value for the parent *IProperty* object, if it has one.

Syntax

.Net

C#	`ISettingValue  RangeMaximum()`
C++	`ISettingValue^ RangeMaximum()`
VB	`Function RangeMaximum As ISettingValue`

ObjC

```
- (P1CaptureCore_SettingValue *) rangeMaximum
```

Return Value

A reference to an *ISettingValue* object that represents the maximum value for the parent *IProperty* object. A NULL reference is returned if the property doesn't have a range limit.

Remarks

If there is a range limit, then both the minimum and maximum range values will be defined. An application can call *HasRange* to test if there is a range limit.

3.16.7 HasValueList

HasValueList returns true if there is a list of values that the parent *IProperty* object can be set to. If true, then *GetValueList* can be used to retrieve the list.

Syntax

.Net

C#	`bool HasValueList()`
C++	`bool HasValueList()`
VB	`Function HasValueList As Boolean`

ObjC

```
- (BOOL) hasValueList
```

Return Value

True if the *ISettingDescriptor* object contains a list of values that the parent *IProperty* object can be set to.

Remarks

If *HasValueList* returns true, the method *IsValueListSelectOnly* can be used to determine if the list is select-only or not.

3.16.8 IsValueListSelectOnly

IsValueListSelectOnly returns true if the parent *IProperty* object can only be set to values from the list returned by *GetValueList*.

Syntax

.Net

C#	`bool IsValueListSelectOnly()`
C++	`bool IsValueListSelectOnly()`
VB	`Function IsValueListSelectOnly As Boolean`

ObjC

```
- (BOOL) isValueListSelectOnly
```

Return Value

True if the *ISettingDescriptor* object contains a list of values that the parent *IProperty* object can be set to, and the *IProperty* object can only be set to values from this list.

Remarks

A list of values can be a list of common or suggested values, or it could be a select-only list of values that must be chosen from. The *IsValueListSelectOnly* method will return true for the latter case.

3.16.9 GetValueList

GetValueList returns an optional list of values that the parent *IProperty* object can be set to.

Syntax

.Net

C#	`ISettingValueList  GetValueList()`
C++	`ISettingValueList^ GetValueList()`
VB	`Function GetValueList As ISettingValueList`

ObjC

```
- (P1CaptureCore_SettingValueList *) getValueList
```

Return Value

A reference to an *ISettingValueList* object that contains *ISettingValue* objects that represent values that the parent *IProperty* object can be set to. A NULL reference or an empty list is returned if no values are defined.

Remarks

The list returned by *GetValueList* could be either a list of suggested values or a list of select-only values. If *IsValueListSelectOnly* returns true, the list is a select-only value list and the parent *IProperty* object can only be set to values from the list. If *IsValueListSelectOnly* returns false, then *IProperty* object can be set to any value that is in range as well as those in the value list. In this case, the value list is only a list of suggested or common values.

3.16.10 ValidateValue

ValidateValue tests whether an *IValueRead* object has a value that is valid for the parent *IProperty* object, and throws an exception if it is not. A value is valid if it is in range or in the value list, if any.

Syntax

.Net

C#	`void ValidateValue( IValueRead  value )`
C++	`void ValidateValue( IValueRead^ value )`
VB	`Sub ValidateValue( value As IValueRead )`

ObjC

```
- (void) validateValue: (P1CaptureCore_ValueRead *) value
```

Parameters

value	A reference to an *IValueRead* object whose value will be tested for validity.

Remarks

An exception is thrown if *value* is not a valid setting for the parent *IProperty* object. Typically this is an out of range exception.

If the value type of *value* is not the same as value type of the *ISettingDescriptor* object (or not compatible with it), then *value* is invalid and an exception is thrown.

If the *ISettingDescriptor* has a value list and *value* matches an entry in the list, then *value* is always valid and no further tests are performed. If it does not match an entry in the list and the list is select-only, then *value* is invalid and an exception is thrown.

If there is a range limit and *value* is outside the minimum and maximum values, then *value* is invalid and an exception is thrown.

3.17 ISettingValueList (P1CaptureCore_SettingValueList)

The *ISettingValueList* class is a list container for *ISettingValue* objects. It is a child object of *ISettingDescriptor*, and inherits from *IChildObject* and *IObjectList*.

Members

`ValueType`	Returns the type of values contained within the list. All values in the list are of the same type.
Inherited from IChildObject	
`Parent`	Returns the parent *ISettingDescriptor* object of this object.
Inherited from IObjectList	
`Size`	Returns the number of *ISettingValue* items in the list.
`IsEmpty`	Returns true if the list is empty.
`First`	Returns a reference to the first *ISettingValue* item in the list.
`Last`	Returns a reference to the last *ISettingValue* item in the list.
`Next`	Returns a reference to the next *ISettingValue* item in the list following a specified *ISettingValue* item already in the list.
`Previous`	Returns a reference to the previous *ISettingValue* item in the list preceding a specified *ISettingValue* item already in the list.
`Insert`	Inserts a new item in front of another specified *ISettingValue* item in the list. Requires insert access rights.
`Remove`	Removes a specified *ISettingValue* item from the list. Requires remove access rights.
`Clear`	Removes all items from the list. Requires remove access rights.
`GetAccess`	Returns the access rights for this list as a bitmask of *EnumListAccess* values.
`HasAccess`	Returns true if the list allows the specified access rights.

3.17.1 ValueType

ValueType returns the type of values contained within the list. All values in the list are of the same type.

Syntax

.Net

C#	`EnumValueType ValueType { get; }`
C++	`property EnumValueType^ ValueType { EnumValueType^ get(); }`
VB	`ReadOnly Property ValueType As EnumValueType`

ObjC

```
- (EnumValueType) valueType
```

Return Value

An *EnumValueType* enumeration value that indicates the value type of the *ISettingValue* objects in the *ISettingValueList* object.

Remarks

The value type of an *ISettingValueList* object doesn’t change after the object is created.

3.18 ISettingValue (P1CaptureCore_SettingValue)

The *ISettingValue* class describes a value that an *IProperty* object can be set to. *ISettingValue* objects are read-only and owned by a parent *ISettingDescriptor* object, which is itself owned by the associated *IProperty* object.

ISettingValue objects are read-only objects, however their values can change dynamically, in response to method calls or events. A setting descriptor change event is sent by the *ICaptureObject* object that is indirectly the parent of the *ISettingValue* object, whenever the object's value changes (see the specific class description for the event ID).

ISettingValue is a child object of *ISettingDescriptor*, and inherits from *IChildObject* and *IValueRead*.

Members

Inherited from IChildObject	
`Parent`	Returns the parent *ISettingDescriptor* object of this object.
Inherited from IValueRead	
`ValueType`	Returns the value type (Boolean, integer, string, etc) of the object.
`IsUndefined`	Returns true if the object's value is undefined.
`GetValue`	Gets the value of the object if the type of the object and the type passed to *GetValue* are compatible. One can always get a string representation for all value types. *GetValue* is available on platforms that support overloading.
`GetValueBool`	Returns the value of the object if its value type is a Boolean.
`GetValueInt32`	Returns the value of the object if its value type is a 32-bit signed integer (or enumeration).
`GetValueUInt32`	Returns the value of the object if its value type is a 32-bit unsigned integer (or enumeration).
`GetValueInt64`	Returns the value of the object if its value type is a 64-bit signed integer.
`GetValueUInt64`	Returns the value of the object if its value type is a 64-bit unsigned integer.
`GetValueFloat64`	Returns the value of the object if its value type is a 64-bit floating point.
`GetValueString`	Returns the value of the object if its value type is a string, or a string representation of the value for all other value types.
`GetValueEnum`	Returns the value of the object if its value type is an enumeration (32-bit signed integer).
`GetValuePoint`	Returns the value of the object if its value type is a point (32-bit signed integer).
`GetValuePointFloat`	Returns the value of the object if its value type is a point (64-bit floating point).

`GetValueArea`	Returns the value of the object if its value type is an area (32-bit signed integer).
`GetValueAreaFloat`	Returns the value of the object if its value type is an area (64-bit floating point).
`GetValueRect`	Returns the value of the object if its value type is a rectangle (32-bit signed integer).
`GetValueRectFloat`	Returns the value of the object if its value type is a rectangle (64-bit floating point).
`Compare`	Compares this object's value to another object of the same value type, returning a signed integer representing if this object is less than, greater than or equal to the other object.

3.19 IRootObject (P1CaptureCore_RootObject)

All CaptureCore class interfaces are derived from *IRootObject*. *IRootObject* represents functionality that is common to all CaptureCore classes. Currently *IRootObject* has no methods and is a place holder for future functionality. It does not exist as an object on its own, and is only accessible via a derived class.

3.20 IChildObject (P1CaptureCore_ChildObject)

The *IChildObject* base class provides common functionality for all CaptureCore objects that are a child of another CaptureCore object. It does not exist as an object on its own, and is only accessible via a derived class.

Members

`Parent`	Returns the parent object of this object.

3.20.1 Parent

Parent returns the parent object of this object.

Syntax

.Net

C#	`ParentType  Parent()`
C++	`ParentType^ Parent()`
VB	`Function Parent As ParentType`

ObjC

```
- (P1CaptureCore_RootObject *) parent
```

Return Value

The parent object of this object. A NULL reference is returned if this object currently has no parent, that is if this object is an orphaned child object.

In development environments where generics or templates are supported, such as .NET, the type of the parent object matches the immediate parent of this object in the data hierarchy. In other environments, such as ObjC, the return type is always a *IRootObject*.

Remarks

If the type of the return value is not directly useful, it can be cast to a valid base class or a derived class of the parent class. Dynamic or type-safe casting should always be used to verify that the parent object is actually of the desired class before performing the type cast.

3.21 IObjectList (P1CaptureCore_ObjectList)

The *IObjectList* base class provides a common set of list container functionality for CaptureCore objects. It does not exist as an object on its own, and is only accessible via a derived class.

Many object lists in CaptureCore dynamically change in response to method calls or events. When an *IObjectList* object is returned from a method call, it is generally a copy of another internally maintained object list. Thus it is a snapshot of the state of the internal list at the time the list was retrieved. This allows the returned list to be used without any problems that may arise from dynamic changes – only the internal object list is changed dynamically. An event is generally sent when an internal list changes, and subsequently the application can retrieve a new copy of the internal list.

Members

`Size`	Returns the number of items in the list.
`IsEmpty`	Returns true if the list is empty.
`First`	Returns a reference to the first item in the list.
`Last`	Returns a reference to the last item in the list.
`Next`	Returns a reference to the next item in the list following a specified item already in the list.
`Previous`	Returns a reference to the previous item in the list preceding a specified item already in the list.
`Insert`	Inserts a new item in front of another specified item in the list. Requires insert access rights.
`Remove`	Removes an item from the list. Requires remove access rights.
`Clear`	Removes all items from the list. Requires remove access rights.
`GetAccess`	Returns the access rights for this list as a bitmask of *EnumListAccess* values.
`HasAccess`	Returns true if the list allows the specified access rights.

3.21.1 Size

Size returns the number of items in the list.

Syntax

.Net

C#	`uint Size()`
C++	`System::UInt32 Size()`
VB	`Function Size As UInteger`

ObjC

```
- (uint32_t) size
```

Return Value

The number of items in the list.

3.21.2 IsEmpty

IsEmpty returns true if the list is empty.

Syntax

.Net

C#	`bool IsEmpty()`
C++	`bool IsEmpty()`
VB	`Function IsEmpty As Boolean`

ObjC

```
- (BOOL) isEmpty
```

Return Value

True if the list is empty, that is if its size is zero.

3.21.3 First

First returns a reference to the first item in the list.

Syntax

.Net

C#	`ItemType  First()`
C++	`ItemType^ First()`
VB	`Function First As ItemType`

ObjC

```
- (P1CaptureCore_RootObject *) first
```

Return Value

A reference to the first item in the list. A NULL reference is returned if the list is empty.

3.21.4 Last

Last returns a reference to the last item in the list.

Syntax

.Net

C#	`ItemType  Last()`
C++	`ItemType^ Last()`
VB	`Function Last As ItemType`

ObjC

```
- (P1CaptureCore_RootObject *) last
```

Return Value

A reference to the list item in the list. A NULL reference is returned if the list is empty.

3.21.5 Next

Next returns a reference to the next item in the list following a specified item already in the list.

Syntax

.Net

C#	`ItemType  Next( ItemType  index )`
C++	`ItemType^ Next( ItemType^ index )`
VB	`Function Next( index As ItemType ) As ItemType`

ObjC

```
- (P1CaptureCore_RootObject *) next: (P1CaptureCore_RootObject *) index
```

Parameters

index A reference to an item in the list.

Return Value

A reference to the next item in the list following the item referred to by *index*. A NULL reference is returned, if is *index* is the last item, or if *index* is not in the list or is itself a NULL reference.

Remarks

Next can be used with *First* to iterate forwards through all the items in the list.

3.21.6 Previous

Previous returns a reference to the previous item in the list preceding a specified item already in the list.

Syntax

.Net

C#	`ItemType  Previous( ItemType  index )`
C++	`ItemType^ Previous( ItemType^ index )`
VB	`Function Previous( index As ItemType ) As ItemType`

ObjC

```
- (P1CaptureCore_RootObject *) previous: (P1CaptureCore_RootObject *) index
```

Parameters

index A reference to an item in the list.

Return Value

A reference to the previous item in the list preceding the item referred to by *index*. A NULL reference is returned, if *index* is the first item, or if *index* is not in the list or is itself a NULL reference.

Remarks

Previous can be used with *Last* to iterate backwards through all the items in the list.

3.21.7 Insert

Insert inserts a new item in front of another specified item in the list. Requires insert access rights.

Syntax

.Net

C#	`bool Insert( ItemType  item, ItemType  before )`

C++	`bool Insert( ItemType^ item, ItemType^ before )`
VB	`Function Insert( item As ItemType, before As ItemType ) As Boolean`

ObjC

```
- (void) insert: (P1CaptureCore_RootObject *) item
                 before: (P1CaptureCore_RootObject *) before
```

Parameters

item	A reference to an item to insert in the list. If *item* is already in the list or is a NULL reference, then *Insert* does nothing.
before	An optional reference to an item already in the list that *item* should be inserted after. If *before* is a NULL reference or not in the list, then *item* is inserted at the end of the list.

Return Value

True if *item* was added to the list, otherwise false. False is returned if *item* is already in the list or is a NULL reference.

Remarks

To call *Insert*, the application must have insert access rights (*kListAccess_Insert*) for the *IObjectList* object, otherwise an exception will be thrown. *HasAccess* can be used to test if the application has specific access rights for the *IObjectList* object.

3.21.8 Remove

Remove removes an item from the list. Requires remove access rights.

Syntax

.Net

C#	`bool Remove( ItemType  item )`
C++	`bool Remove( ItemType^ item )`
VB	`Function Remove( item As ItemType ) As Boolean`

ObjC

```
- (void) remove: (P1CaptureCore_RootObject *) item
```

Parameters

item	A reference to an item in the list to remove. If *item* is not in the list or is a NULL reference, then *Remove* does nothing.

Return Value

True if the item was in the list and thus removed, otherwise false.

Remarks

To call *Remove*, the application must have remove access rights (*kListAccess_Remove*) for the *IObjectList* object, otherwise an exception will be thrown. *HasAccess* can be used to test if the application has specific access rights for the *IObjectList* object.

3.21.9 Clear

Clear removes all items from the list. Requires remove access rights.

Syntax

.Net

C#	`void Clear()`
C++	`void Clear()`
VB	`Sub Clear`

ObjC

```
- (void) clear
```

Remarks

To call *Clear*, the application must have remove access rights (*kListAccess_Remove*) for the *IObjectList* object, otherwise an exception will be thrown. *HasAccess* can be used to test if the application has specific access rights for the *IObjectList* object.

3.21.10 GetAccess

GetAccess returns the access rights for this list as a bitmask of *EnumListAccess* values.

Syntax

.Net

C#	`EnumListAccess GetAccess()`
C++	`EnumListAccess GetAccess()`
VB	`Function GetAccess As EnumListAccess`

ObjC

```
- (uint32_t) getAccess
```

Return Value

A bitmask of *EnumListAccess* enumeration values indicating which access rights the application has for the *IListObject* object.

Remarks

Different *IListObject* objects throughout CaptureCore will allow the application different access rights to the list. Some lists are read-only, while others can be removed from but not inserted to, and some allow full access.

3.21.11 HasAccess

HasAccess returns true if the list allows the specified access rights.

Syntax

.Net

C#	`bool HasAccess( EnumListAccess access )`
C++	`bool HasAccess( EnumListAccess access )`
VB	`Function HasAccess( access As EnumListAccess ) As Boolean`

ObjC

```
- (BOOL) hasAccess: (uint32_t) access
```

Parameters

access	A bitmask combination of *EnumListAccess* enumeration values of the access rights to test for.

Return Value

True if the list allows all the access rights specified in *access*, otherwise false.

Remarks

Different *IListObject* objects throughout CaptureCore will allow the application different access rights to the list. Some lists are read-only, while others can be removed from but not inserted to, and some allow full access.

3.22 IValueRead (P1CaptureCore_ValueRead)

The *IValueRead* base class provides a common set of functionality for objects that contain a value (number, string, Boolean, etc). *IValueRead* only provides reading functionality, the *IValueWrite* class provides writing functionality for objects where the value can be modified. *IValueRead* does not exist as an object on its own, and is only accessible via a derived class.

An *IValueRead* object can contain an undefined value. If the *IValueRead* method *IsUndefined* returns true, the object's value is undefined and cannot be interpreted by the application. Applications should not call any of the get value methods if the *IValueRead* object indicates that its value is undefined.

Members

`ValueType`	Returns the value type (Boolean, integer, string, etc) of the object.
`IsUndefined`	Returns true if the object's value is undefined.
`GetValue`	Gets the value of the object if the type of the object and the type passed to *GetValue* are compatible. One can always get a string representation for all value types. *GetValue* is available on platforms that support overloading.
`GetValueBool`	Returns the value of the object if its value type is a Boolean.
`GetValueInt32`	Returns the value of the object if its value type is a 32-bit signed integer (or enumeration).
`GetValueUInt32`	Returns the value of the object if its value type is a 32-bit unsigned integer (or enumeration).
`GetValueInt64`	Returns the value of the object if its value type is a 64-bit signed integer.
`GetValueUInt64`	Returns the value of the object if its value type is a 64-bit unsigned integer.
`GetValueFloat64`	Returns the value of the object if its value type is a 64-bit floating point.
`GetValueString`	Returns the value of the object if its value type is a string, or a string representation of the value for all other value types.
`GetValueEnum`	Returns the value of the object if its value type is an enumeration (32-bit signed integer).
`GetValuePoint`	Returns the value of the object if its value type is a point (32-bit signed integer).
`GetValuePointFloat`	Returns the value of the object if its value type is a point (64-bit floating point).
`GetValueArea`	Returns the value of the object if its value type is an area (32-bit signed integer).
`GetValueAreaFloat`	Returns the value of the object if its value type is an area (64-bit floating point).
`GetValueRect`	Returns the value of the object if its value type is a rectangle (32-bit signed integer).

`GetValueRectFloat`	Returns the value of the object if its value type is a rectangle (64-bit floating point).
`Compare`	Compares this object's value to another object of the same value type, returning a signed integer representing if this object is less than, greater than or equal to the other object.

3.22.1 ValueType

ValueType returns the value type (Boolean, integer, string, etc) of the object.

Syntax

.Net

C#	`EnumValueType ValueType { get; }`
C++	`property EnumValueType^ ValueType { EnumValueType^ get(); }`
VB	`ReadOnly Property ValueType As EnumValueType`

ObjC

```
- (EnumValueType) valueType
```

Return Value

An *EnumValueType* enumeration value that indicates the value type of the *IValueRead* object.

Remarks

The value type of an *IValueRead* object doesn't change after the object is created.

3.22.2 IsUndefined

IsUndefined returns true if the object's value is undefined.

An *IValueRead* object can contain an undefined value, that is a value that is not specified. This allows CaptureCore to report values that are unknown, invalid, or in some other way indeterminate. Such a value cannot be interpreted by the application.

Syntax

.Net

C#	`bool IsUndefined()`
C++	`bool IsUndefined()`
VB	`Function IsUndefined As Boolean`

ObjC

```
- (BOOL) isUndefined
```

Return Value

True if the value of the *IValueRead* object is undefined, otherwise false.

Remarks

If *IsUndefined* returns true, an application should not call any of the get value methods nor try to interpret the object's value.

IsUndefined always returns true if the value type of the object is *kValueType_Undefined*. However, it can also be undefined for all other value types.

3.22.3 Get Value Methods

There are several different methods that retrieve the value of the *IValueRead* object. *GetValue* gets the value of the object if the type of the object and the type passed to *GetValue* are compatible. *GetValue* is available on platforms that support overloading. *GetValueBool*, *GetValueInt32*, *GetValueUInt32*, *GetValueInt64*, *GetValueUInt64*, *GetValueFloat64*, *GetValueString*, and *GetValueEnum* return the value of the object if its value type matches the method called.

One can always get a string representation for all value types by calling *GetValue* with a string argument or *GetValueString*. In the case of an enumeration value type, in addition to *GetValueEnum*, one can also call *GetValue* with a 32-bit signed or unsigned integer, or *GetValueInt32* or *GetValueUInt32*.

The following table summarizes which methods can be called for each value type.

`kValueType_Undefined`	None
`kValueType_Bool`	*GetValue*, *GetValueString*, *GetValueBool*
`kValueType_Int32`	*GetValue*, *GetValueString*, *GetValueInt32*
`kValueType_UInt32`	*GetValue*, *GetValueString*, *GetValueUInt32*
`kValueType_Int64`	*GetValue*, *GetValueString*, *GetValueInt64*
`kValueType_UInt64`	*GetValue*, *GetValueString*, *GetValueUInt64*
`kValueType_Float64`	*GetValue*, *GetValueString*, *GetValueFloat64*
`kValueType_String`	*GetValue*, *GetValueString*
`kValueType_Enum`	*GetValue*, *GetValueString*, *GetValueEnum*, *GetValueInt32*, *GetValueUInt32*
`kValueType_Point`	*GetValue*, *GetValueString*, *GetValuePoint*
`kValueType_PointFloat`	*GetValue*, *GetValueString*, *GetValuePointFloat*
`kValueType_Area`	*GetValue*, *GetValueString*, *GetValueArea*
`kValueType_AreaFloat`	*GetValue*, *GetValueString*, *GetValueAreaFloat*
`kValueType_Rect`	*GetValue*, *GetValueString*, *GetValueRect*
`kValueType_RectFloat`	*GetValue*, *GetValueString*, *GetValueRectFloat*

Syntax

.Net

C#	`void GetValue( out bool value )`
	`void GetValue( out int value )`
	`void GetValue( out uint value )`
	`void GetValue( out long value )`
	`void GetValue( out ulong value )`
	`void GetValue( out double value )`
	`void GetValue( out string value )`
	`void GetValue( out System.Drawing.Point value )`
	`void GetValue( out System.Drawing.PointF value )`
	`void GetValue( out System.Drawing.Size value )`
	`void GetValue( out System.Drawing.SizeF value )`
	`void GetValue( out System.Drawing.Rectangle value )`
	`void GetValue( out System.Drawing.RectangleF value )`

	bool GetValueBool()
	int GetValueInt32()
	uint GetValueUInt32()
	long GetValueInt64()
	ulong GetValueUInt64()
	double GetValueFloat64()
	string GetValueString()
	int GetValueEnum()
	System.Drawing.Point GetValuePoint()
	System.Drawing.PointF GetValuePointFloat()
	System.Drawing.Size GetValueArea()
	System.Drawing.SizeF GetValueAreaFloat()
	System.Drawing.Rectangle GetValueRectangle()
	System.Drawing.RectangleF GetValueRectangleFloat()
C++	void GetValue([System::Runtime::InteropServices::Out] bool% value)
	void GetValue([System::Runtime::InteropServices::Out] System::Int32% value)
	void GetValue([System::Runtime::InteropServices::Out] System::UInt32% value)
	void GetValue([System::Runtime::InteropServices::Out] System::Int64% value)
	void GetValue([System::Runtime::InteropServices::Out] System::UInt64% value)
	void GetValue([System::Runtime::InteropServices::Out] double% value)
	void GetValue([System::Runtime::InteropServices::Out] System::String^% value)
	void GetValue([System::Runtime::InteropServices::Out] System::Drawing::Point^% value)
	void GetValue([System::Runtime::InteropServices::Out] System::Drawing::PointF^% value)
	void GetValue([System::Runtime::InteropServices::Out] System::Drawing::Size^% value)
	void GetValue([System::Runtime::InteropServices::Out] System::Drawing::SizeF^% value)
	void GetValue([System::Runtime::InteropServices::Out] System::Drawing::Rectangle^% value)
	void GetValue([System::Runtime::InteropServices::Out] System::Drawing::RectangleF^% value)
	bool GetValueBool()
	System::Int32 GetValueInt32()
	System::UInt32 GetValueUInt32()
	System::Int64 GetValueInt64()
	System::UInt64 GetValueUInt64()
	double GetValueFloat64()
	System::String^ GetValueString()
	System::Int32 GetValueEnum()
	System::Drawing::Point^ GetValuePoint()
	System::Drawing::PointF^ GetValuePointFloat()

	System::Drawing::Size^ GetValueArea()
	System::Drawing::SizeF^ GetValueAreaFloat()
	System::Drawing::Rectangle^ GetValueRect()
	System::Drawing::RectangleF^ GetValueRectFloat()
VB	Sub GetValue(ByRef value As Boolean)
	Sub GetValue(ByRef value As Integer)
	Sub GetValue(ByRef value As UInteger)
	Sub GetValue(ByRef value As Long)
	Sub GetValue(ByRef value As ULong)
	Sub GetValue(ByRef value As Double)
	Sub GetValue(ByRef value As String)
	Sub GetValue(ByRef value As System.Drawing.Point)
	Sub GetValue(ByRef value As System.Drawing.PointF)
	Sub GetValue(ByRef value As System.Drawing.Size)
	Sub GetValue(ByRef value As System.Drawing.SizeF)
	Sub GetValue(ByRef value As System.Drawing.Rectangle)
	Sub GetValue(ByRef value As System.Drawing.RectangleF)
	Function GetValueBool As Boolean
	Function GetValueInt32 As Integer
	Function GetValueUInt32 As UInteger
	Function GetValueInt64 As Long
	Function GetValueUInt64 As ULong
	Function GetValueFloat64 As Double
	Function GetValueString As String
	Function GetValueEnum As Integer
	Function GetValuePoint As System.Drawing.Point
	Function GetValuePointFloat As System.Drawing.PointF
	Function GetValueArea As System.Drawing.Size
	Function GetValueAreaFloat As System.Drawing.SizeF
	Function GetValueRect As System.Drawing.Rectangle
	Function GetValueRectFloat As System.Drawing.RectangleF

ObjC

- (BOOL) getValueBool
- (int32_t) getValueInt32
- (uint32_t) getValueUInt32
- (int64_t) getValueInt64
- (uint64_t) getValueUInt64
- (double) getValueFloat64
- (NSString *) getValueString
- (int32_t) getValueEnum
- (NSValue *) getValuePoint
- (NSValue *) getValuePointFloat
- (NSValue *) getValueArea
- (NSValue *) getValueAreaFloat
- (NSValue *) getValueRect
- (NSValue *) getValueRectFloat

Parameters

value	*GetValue* only. A reference to language specific type that is compatible with the value type of this *IValueRead* object. The value of this *IValueRead* object is returned via this parameter.

Return Value

A language specific type that is appropriate for the value type requested and that represents the value of this *IValueRead* object.

Remarks

All the get value methods (except *GetValue* with a string parameter or *GetValueString*) throw an exception if the requested value type is not compatible with the value type of this *IValueRead* object.

3.22.4 Compare

Compare compares this object's value to another object of the same value type, returning a signed integer representing if this object is less than, greater than or equal to the other object.

Syntax

.Net

C#	`int Compare( IValueRead otherValue )`
C++	`System::Int32 Compare( IValueRead^ otherValue )`
VB	`Function Compare( otherValue As IValueRead ) As Integer`

ObjC

```
- (NSComparisonResult) compare: (P1CaptureCore_ValueRead *) otherValue
```

Parameters

otherValue	A reference to another *IValueRead* object to compare with this object.

Return Value

An integer value that is zero if this object is equal to *otherValue*, a positive value if this object is greater than *otherValue*, and a negative value if this object is less than *otherValue*.

Remarks

Compare throws an exception if the value type of *otherValue* is not the same or not compatible with the value type of this object.

3.23 IValueWrite (P1CaptureCore_ValueWrite)

The *IValueWrite* base class provides a common set of functionality for objects that contain a value (number, string, Boolean, etc). *IValueWrite* inherits from *IValueRead*, and provides both writing and reading functionality (via inherited *IValueRead* methods). It does not exist as an object on its own, and is only accessible via a derived class.

IValueWrite objects can be read-only, despite having methods for changing their value. This is because some methods may return both types of values, a writeable or read-only value. In these cases, *IValueWrite* objects are returned, but the *IValueWrite* method *IsReadOnly* will return true. *IsReadOnly* should always be checked before calling *IValueWrite* methods.

Members

`IsReadOnly`	Returns true if the object is a read-only object. Set value methods may not be called on a read-only object.
`SetValue`	Sets the value of the object if the type of the object and the type passed to *SetValue* are compatible. *SetValue* is available on platforms that support overloading.
`SetValueBool`	Sets the value of the object if its value type is a Boolean.
`SetValueInt32`	Sets the value of the object if its value type is a 32-bit signed integer (or enumeration).
`SetValueUInt32`	Sets the value of the object if its value type is a 32-bit unsigned integer (or enumeration).
`SetValueInt64`	Sets the value of the object if its value type is a 64-bit signed integer.
`SetValueUInt64`	Sets the value of the object if its value type is a 64-bit unsigned integer.
`SetValueFloat64`	Sets the value of the object if its value type is a 64-bit floating point.
`SetValueString`	Sets the value of the object if its value type is a string.
`SetValueEnum`	Sets the value of the object if its value type is an enumeration (32-bit signed integer).
`SetValuePoint`	Sets the value of the object if its value type is a point (32-bit signed integer).
`SetValuePointFloat`	Sets the value of the object if its value type is a point (64-bit floating point).
`SetValueArea`	Sets the value of the object if its value type is an area (32-bit signed integer).
`SetValueAreaFloat`	Sets the value of the object if its value type is an area (64-bit floating point).
`SetValueRect`	Sets the value of the object if its value type is a rectangle (32-bit signed integer).
`SetValueRectFloat`	Sets the value of the object if its value type is a rectangle (64-bit floating point).

Inherited from IValueRead	
`ValueType`	Returns the value type (Boolean, integer, string, etc) of the object.
`IsUndefined`	Returns true if the object's value is undefined.
`GetValue`	Gets the value of the object if the type of the object and the type passed to *GetValue* are compatible. One can always get a string representation for all value types. *GetValue* is available on platforms that support overloading.
`GetValueBool`	Returns the value of the object if its value type is a Boolean.
`GetValueInt32`	Returns the value of the object if its value type is a 32-bit signed integer (or enumeration).
`GetValueUInt32`	Returns the value of the object if its value type is a 32-bit unsigned integer (or enumeration).
`GetValueInt64`	Returns the value of the object if its value type is a 64-bit signed integer.
`GetValueUInt64`	Returns the value of the object if its value type is a 64-bit unsigned integer.
`GetValueFloat64`	Returns the value of the object if its value type is a 64-bit floating point.
`GetValueString`	Returns the value of the object if its value type is a string, or a string representation of the value for all other value types.
`GetValueEnum`	Returns the value of the object if its value type is an enumeration (32-bit signed integer).
`GetValuePoint`	Returns the value of the object if its value type is a point (32-bit signed integer).
`GetValuePointFloat`	Returns the value of the object if its value type is a point (64-bit floating point).
`GetValueArea`	Returns the value of the object if its value type is an area (32-bit signed integer).
`GetValueAreaFloat`	Returns the value of the object if its value type is an area (64-bit floating point).
`GetValueRect`	Returns the value of the object if its value type is a rectangle (32-bit signed integer).
`GetValueRectFloat`	Returns the value of the object if its value type is a rectangle (64-bit floating point).
`Compare`	Compares this object's value to another object of the same value type, returning a signed integer representing if this object is less than, greater than or equal to the object.

3.23.1 IsReadOnly

IsReadOnly returns true if the object is a read-only object. Set value methods may not be called on a read-only object.

Syntax

.Net

C#	`bool IsReadOnly()`
C++	`bool IsReadOnly()`
VB	`Function IsReadOnly As Boolean`

ObjC

```
- (BOOL) isReadOnly
```

Return Value

True if the *IValueWrite* object is read-only.

3.23.2 Set Value Methods

There are several different methods that set the value of the *IValueWrite* object. *SetValue* sets the value of the object if the type of the object and the type passed to *SetValue* are compatible. *SetValue* is available on platforms that support overloading. *SetValueBool*, *SetValueInt32*, *SetValueUInt32*, *SetValueInt64*, *SetValueUInt64*, *SetValueFloat64*, *SetValueString*, and *SetValueEnum* set the value of the object if its value type matches the method called.

In the case of an enumeration value type, in addition to *SetValueEnum*, one can also call *SetValue* with a 32-bit signed or unsigned integer, or *SetValueInt32* or *SetValueUInt32*.

The following table summarizes which methods can be called for each value type.

`kValueType_Undefined`	None
`kValueType_Bool`	*SetValue, SetValueBool*
`kValueType_Int32`	*SetValue, SetValueInt32*
`kValueType_UInt32`	*SetValue, SetValueUInt32*
`kValueType_Int64`	*SetValue, SetValueInt64*
`kValueType_UInt64`	*SetValue, SetValueUInt64*
`kValueType_Float64`	*SetValue, SetValueFloat64*
`kValueType_String`	*SetValue, SetValueString*
`kValueType_Enum`	*SetValue, SetValueEnum, SetValueInt32, SetValueUInt32*
`kValueType_Point`	*SetValue, SetValuePoint*
`kValueType_PointFloat`	*SetValue, SetValuePointFloat*
`kValueType_Area`	*SetValue, SetValueArea*
`kValueType_AreaFloat`	*SetValue, SetValueAreaFloat*
`kValueType_Rect`	*SetValue, SetValueRect*
`kValueType_RectFloat`	*SetValue, SetValueRectFloat*

Syntax

.Net

C#	`void SetValue( bool value )`
	`void SetValue( int value )`
	`void SetValue( uint value )`
	`void SetValue( long value )`
	`void SetValue( ulong value )`

	void SetValue(double value)
	void SetValue(string value)
	void SetValue(System.Drawing.Point value)
	void SetValue(System.Drawing.PointF value)
	void SetValue(System.Drawing.Size value)
	void SetValue(System.Drawing.SizeF value)
	void SetValue(System.Drawing.Rectangle value)
	void SetValue(System.Drawing.RectangleF value)
	void SetValue(IValueRead value)
	void SetValueBool(bool value)
	void SetValueInt32(int value)
	void SetValueUInt32(uint value)
	void SetValueInt64(long value)
	void SetValueUInt64(ulong value)
	void SetValueFloat64(double value)
	void SetValueString(string value)
	void SetValueEnum(int value)
	void SetValuePoint(System.Drawing.Point value)
	void SetValuePointFloat(System.Drawing.PointF value)
	void SetValueArea(System.Drawing.Size value)
	void SetValueAreaFloat(System.Drawing.SizeF value)
	void SetValueRect(System.Drawing.Rectangle value)
	void SetValueRectFloat(System.Drawing.RectangleF value)
C++	void SetValue(bool value)
	void SetValue(System::Int32 value)
	void SetValue(System::UInt32 value)
	void SetValue(System::Int64 value)
	void SetValue(System::UInt64 value)
	void SetValue(double value)
	void SetValue(System::String^ value)
	void SetValue(System::Drawing::Point^ value)
	void SetValue(System::Drawing::PointF^ value)
	void SetValue(System::Drawing::Size^ value)
	void SetValue(System::Drawing::SizeF^ value)
	void SetValue(System::Drawing::Rectangle^ value)
	void SetValue(System::Drawing::RectangleF^ value)
	void SetValue(IValueRead^ value)
	void SetValueBool(bool value)
	void SetValueInt32(System::Int32 value)
	void SetValueUInt32(System::UInt32 value)
	void SetValueInt64(System::Int64 value)
	void SetValueUInt64(System::UInt64 value)
	void SetValueFloat64(double value)
	void SetValueString(System::String^ value)
	void SetValueEnum(System::Int32 value)
	void SetValuePoint(System::Drawing::Point^ value)
	void SetValuePointFloat(System::Drawing::PointF^ value)

	void SetValueArea(System::Drawing::Size^ value)
	void SetValueAreaFloat(System::Drawing::SizeF^ value)
	void SetValueRect(System::Drawing::Rectangle^ value)
	void SetValueRectFloat(System::Drawing::RectangleF^ value)
VB	Sub SetValue(value As Boolean)
	Sub SetValue(value As Integer)
	Sub SetValue(value As UInteger)
	Sub SetValue(value As Long)
	Sub SetValue(value As ULong)
	Sub SetValue(value As Double)
	Sub SetValue(value As String)
	Sub SetValue(value As System.Drawing.Point)
	Sub SetValue(value As System.Drawing.PointF)
	Sub SetValue(value As System.Drawing.Size)
	Sub SetValue(value As System.Drawing.SizeF)
	Sub SetValue(value As System.Drawing.Rectangle)
	Sub SetValue(value As System.Drawing.RectangleF)
	Sub SetValue(value As IValueRead)
	Sub SetValueBool(value As Boolean)
	Sub SetValueInt32(value As Integer)
	Sub SetValueUInt32(value As UInteger)
	Sub SetValueInt64(value As Long)
	Sub SetValueUInt64(value As ULong)
	Sub SetValueFloat64(value As Double)
	Sub SetValueString(value As String)
	Sub SetValueEnum(value As Integer)
	Sub SetValuePoint(value As System.Drawing.Point)
	Sub SetValuePointFloat(value As System.Drawing.PointF)
	Sub SetValueArea(value As System.Drawing.Size)
	Sub SetValueAreaFloat(value As System.Drawing.SizeF)
	Sub SetValueRect(value As System.Drawing.Rectangle)
	Sub SetValueRectFloat(value As System.Drawing.RectangleF)

ObjC

- (void) setValue: (P1CaptureCore_ValueRead *) value
- (void) setValueBool: (BOOL) value
- (void) setValueInt32: (int32_t) value
- (void) setValueUInt32: (uint32_t) value
- (void) setValueInt64: (int64_t) value
- (void) setValueUInt64: (uint64_t) value
- (void) setValueFloat64: (double) value
- (void) setValueString: (NSString *) value
- (void) setValueEnum: (int32_t) value
- (void) setValuePoint: (NSValue *) value
- (void) setValuePointFloat: (NSValue *) value
- (void) setValueArea: (NSValue *) value
- (void) setValueAreaFloat: (NSValue *) value
- (void) setValueRect: (NSValue *) value

```
- (void) setValueRectFloat:  (NSValue *)  value
```

Parameters

value	A language specific type that is compatible with the value type of this *IValueWrite* object, and that contains the value to set.

Remarks

All the set value methods throw an exception if value type of *value* is not compatible with the value type of this *IValueRead* object, or if the *IValueWrite* object is read-only.

3.24 IErrorSource (P1CaptureCore_ErrorSource)

The *IErrorSource* base class provides a common set of functionality for classes that can queue errors. It does not exist as an object on its own, and is only accessible via a derived class.

Most classes throw exceptions when a method call encounters an error. Classes derived from *IErrorSource*, such as all *ICaptureObject* derived classes (*ICaptureProvider*, *ICamera*, and *ICaptureImage*), may choose to queue an error instead of throwing an exception. This is typically done when an error occurs outside of a method call, such as in a background thread. Such classes will generally still throw exceptions when an error occurs in a method call.

IErrorSource objects maintain an internal queue of *IErrorObject* objects, and post a *kEventId_Error* event whenever a new *IErrorObject* object is added to the queue. Applications can call the *IErrorSource* method *GetError* to retrieve the next error object in the queue. Note that the error queue can grow indefinitely if *GetError* is not called to remove errors from the queue.

IErrorSource is a parent to *IErrorObject* objects.

Members

`GetError`	Returns the next *IErrorObject* object, if any, for the *IErrorSource* object.

Events

General `(EnumGeneralEventId)`	
`kEventId_Error`	An error has occurred on a background thread. Indicates that a new *IErrorObject* object has been queued by this object.

3.24.1 GetError

GetError returns the next *IErrorObject* object, if any, for the *IErrorSource* object.

Syntax

.Net

C#	`IErrorObject  GetError()`
C++	`IErrorObject^ GetError()`
VB	`Function GetError As IErrorObject`

ObjC

```
- (P1CaptureCore_ErrorObject *) getError
```

Return Value

A reference to an *IErrorObject* object that contains the next error in the error queue of the *IErrorSource* object. A NULL reference is returned if there are no errors in the error queue.

Remarks

GetError removes the error from the error queue. So subsequent calls to *GetError* will return new errors, or a NULL reference if there are no queued errors when the method is called.

3.24.2 kEventId_Error

This event is posted by the *IErrorSource* object when an *IErrorObject* object has been added to the error list of the *IErrorSource* object.

Arguments

None

3.25 IErrorObject (P1CaptureCore_ErrorObject)

The *IErrorObject* class contains information about an error encountered by an *IErrorSource* derived class. *IErrorObject* objects are generally created by an *IErrorSource* object whenever an error is encountered that can not be reported by throwing an exception. *IErrorObjects* are read-only objects.

IErrorObject is a child object of *IErrorSource*, and inherits from *IChildObject*.

Members

`Type`	Returns the error type for the error, which is generally one of the *EnumErrorType* enumeration values, but can also be other values.
`Number`	Returns the error number for the error, that is unique for a specific error type. If the error type is *kErrorType_CaptureCore*, the error number corresponds to an *EnumCaptureCoreError* enumeration value.
`TypeName`	Returns a string describing the error type.
`Description`	Returns a string describing the error.
`Detail`	Returns an optional string containing additional details about the error.
Inherited from IChildObject	
`Parent`	Returns the parent *IErrorSource* object of this object.

3.25.1 Type

Type returns the error type for the error, which is generally one of the *EnumErrorType* enumeration values, but can also be other values.

The error type represents a category or source of an error. There can be many error types, and the most common are defined by the *EnumErrorType* enumeration. The error number returned by the *Number* method is unique for a specific error type, but errors of different error types may use the same error number. Therefore, an application should always check the error type before interpreting the error number.

Syntax

.Net

C#	`property uint Type;`
C++	`property System::UInt32 Type` `{ System::UInt32 get(); }`
VB	`ReadOnly Property Type As UInteger`

ObjC

```
- (uint32) type
```

Return Value

An numerical value representing the error type of the *IErrorObject* object. Generally, the value will be one of the *EnumErrorType* enumeration values, but can be other values as well.

Remarks

A specific error is defined by both the error type and the error number. The error type of an *IErrorObject* object doesn't change after the object is created.

3.25.2 Number

Number returns the error number for the error, that is unique for a specific error type.

Syntax

.Net

C#	`property uint Number;`
C++	`property System::UInt32 Number` `{ System::UInt32 get(); }`
VB	`ReadOnly Property Number As UInteger`

ObjC

```
- (uint32_t) number
```

Return Value

A numerical value representing the error number of the *IErrorObject* object. The error number is unique only for a specific error type.

Remarks

A specific error is defined by both the error type and the error number. If the error type is *kErrorType_CaptureCore*, the error number corresponds to an *EnumCaptureCoreError* enumeration value. The error number of an *IErrorObject* object doesn't change after the object is created.

3.25.3 TypeName

TypeName returns a string describing the error type.

Syntax

.Net

C#	`string TypeName { get; }`
C++	`property System::String^ TypeName` `{ System::String^ get(); }`
VB	`ReadOnly Property TypeName As String`

ObjC

```
- (NSString *) typeName
```

Return Value

A string containing the name of the error type returned by the method *Type*. A NULL reference or an empty string can be returned, if no type name is available, but this is very unlikely.

Remarks

This method can be used to display a description of the type of error, even error types not defined by the *EnumErrorType* enumeration.

The type name of an *IErrorObject* object doesn't change after the object is created.

3.25.4 Description

Description returns a string describing the error.

Syntax

.Net

C#	`string Description { get; }`
C++	`property System::String^ Description` `{ System::String^ get(); }`
VB	`ReadOnly Property Description As String`

ObjC

```
- (NSString *) description
```

Return Value

A string describing the error. The string is generally the same for the error type and error number of the *IErrorObject* object. A NULL reference or an empty string can be returned, if no description is available, but this is very unlikely.

Remarks

The description string of an *IErrorObject* object doesn't change after the object is created.

3.25.5 Detail

Detail returns an optional string containing additional details about the error.

Syntax

.Net

C#	`string Detail { get; }`
C++	`property System::String^ Detail` `{ System::String^ get(); }`
VB	`ReadOnly Property Detail As String`

ObjC

```
- (NSString *) detail
```

Return Value

A string describing additional details about the error, such as filename, parameter, and so on. A NULL reference or an empty string will be returned if there is no additional details for the error.

Remarks

The detail string of an *IErrorObject* object doesn't change after the object is created.

3.26 IEventSource (P1CaptureCore_EventSource)

The *IEventSource* base class provides a common set of functionality for classes that can post events to event receivers (*IEventReceiver* derived classes). It does not exist as an object on its own, and is only accessible via a derived class.

Applications can create an event receiver by implementing an *IEventReceiver* derived class. Instances of these application created event receivers can subscribe or unsubscribe to events posted by an *IEventSource* object. This is done by adding or removing an *IEventReceiver* object to the *IEventSource* object's internal receiver list via the methods *AddReceiver* or *RemoveReceiver*.

IEventSource is a parent to *IEventObject* objects.

Members

`AddReceiver`	Adds an *IEventReceiver* object and subscribes to specified events from the *IEventSource* object.
`RemoveReceiver`	Removes a previously added *IEventReceiver* object, unsubscribing to specified events from the *IEventSource* object.

Events

General `(EnumGeneralEventId)`	
`kEventId_All`	Used for subscribing or unsubscribing to all events via *AddReceiver* or *RemoveReceiver*.

3.26.1 AddReceiver

AddReceiver adds an *IEventReceiver* object to the receiver list of the *IEventSource* object, and subscribes it to specified events. This method can be called multiple times to subscribe the same *IEventReceiver* object to multiple events, or it can subscribe to the event ID *kEventId_All* to receive all events.

Syntax

.Net

C#	`void AddReceiver( IEventReceiver receiver, System.IntPtr pContext )`
	`void AddReceiver( uint eventID, IEventReceiver receiver, System.IntPtr pContext )`
C++	`void AddReceiver( IEventReceiver^ receiver, System::IntPtr pContext )`
	`void AddReceiver( System::UInt32 eventID, IEventReceiver^ receiver, System::IntPtr pContext )`
VB	`Sub AddReceiver( receiver As IEventReceiver, pContext As System.IntPtr )`
	`Sub AddReceiver( eventID As UInteger, receiver As IEventReceiver, pContext As System.IntPtr )`

ObjC

```
- (void) addReceiver: (id) receiver selector: (SEL) aSelector
                      eventID: (uint32_t) eventID
                      context: (void *) pContext
```

Parameters

receiver An instance of an object that has implemented *IEventReceiver*.

eventID	The event ID of the event to subscribe *receiver* to. Multiple calls can be made for different event IDs. Pass the event ID *kEventId_All* to subscribe to all events.
pContext	An optional pointer parameter that is passed to the *IEventReceiver* method *OnEvent*, when the specified events are delivered. Multiple calls can be made with different *pContext* values. Pass a NULL reference if not needed.
aSelector	[ObjC only] A selector for which method to call on *receiver* when delivering an event. The method must take the same parameters as the *OnEvent* method.

Remarks

A call to *AddReceiver* is ignored if called more than once with the same parameters as a previous call. If all but the *pContext* parameter is the same, then the specified events will be delivered multiple times to *receiver*, once for each unique *pContext* value.

When an application no longer needs to receive events from the *IEventSource* object, for each call to *AddReceiver* that was made, a matching call to *RemoveReceiver*, with the exact same parameters (including *pContext*), should be performed.

If a *receiver* is subscribed to both *kEventId_All* as well as other events, then the other events will be delivered twice to the receiver. Matching calls to *RemoveReceiver* should still be made for each call to *AddReceiver*.

3.26.2 RemoveReceiver

RemoveReceiver removes a previously added *IEventReceiver* object from the receiver list of the *IEventSource* object, unsubscribing to specified events.

When an application no longer needs to receive events from the *IEventSource* object, a matching call to *RemoveReceiver*, with the exact same parameters (including *pContext*), should be performed, for each call to *AddReceiver* that was made.

Syntax

.Net

C#	`void RemoveReceiver( IEventReceiver receiver, System.IntPtr pContext )`
	`void RemoveReceiver( uint eventID, IEventReceiver receiver, System.IntPtr pContext )`
C++	`void RemoveReceiver( IEventReceiver^ receiver, System::IntPtr pContext )`
	`void RemoveReceiver( System::UInt32 eventID, IEventReceiver^ receiver, System::IntPtr pContext )`
VB	`Sub RemoveReceiver( receiver As IEventReceiver, pContext As System.IntPtr )`
	`Sub RemoveReceiver( eventID As UInteger, receiver As IEventReceiver, pContext As System.IntPtr )`

ObjC

```
- (void) removeReceiver: (id) receiver selector: (SEL) aSelector
                 eventID: (uint32_t) eventID
                 context: (void *) pContext
```

Parameters

receiver	The instance of the *IEventReceiver* object that was passed to a previous call to *AddReceiver*.
eventID	The event ID of the event to unsubscribe *receiver* from that was passed to a previous call to *AddReceiver*. Multiple calls must be made for each event ID that was passed to *AddReceiver*. Pass *kEventId_All* to unsubscribe to all events.
pContext	The pointer parameter that was passed to a previous call to *AddReceiver*. Multiple calls must be made for each *pContext* value that was passed to *AddReceiver*.
aSelector	[ObjC only] The selector that was passed to a previous call to *AddReceiver*.

Remarks

All parameters must match a previous call to *AddReceiver*, otherwise the call to *RemoveReceiver* is ignored. The exception is *eventID*, which may be set to *kEventID_All* to unsubscribe all previous calls to *AddReceiver* that match the other remaining parameters.

The *IEventReceiver* object is only removed from the *IEventSource* object's receiver list, if *RemoveReceiver* calls have been made that match every *AddReceiver* call for *receiver*.

If *RemoveReceiver* is called at the same time an event is being delivered to the *OnEvent* method of *receiver*, the call to *RemoveReceiver* will wait until *OnEvent* is completed. This prevents *RemoveReceiver* from returning before it can guarantee that *OnEvent* is done processing the specified events.

It is completely safe to call *RemoveReceiver* directly from *OnEvent*. However, it is important that *OnEvent* does not wait on any threads, or make calls that wait on any threads, that could directly or indirectly call *RemoveReceiver* for the *IEventReceiver* object. Otherwise, a deadlock will occur.

3.27 IEventReceiver (P1CaptureCore_EventReceiver)

The *IEventReceiver* interface class specifies the methods that need to be implemented by an application defined class in order to receive events from an *IEventSource* object. It provides no functionality of its own and is only an interface specification.

Applications that wish to receive events from an *IEventSource* object can define as many classes as they like to implement the *IEventReceiver* interface. One or more instances of these *IEventReceiver* derived classes can be added to each *IEventSource* object or to multiple *IEventSource* objects. *IEventReceiver* objects can subscribe or unsubscribe to events by calling the *IEventSource* methods *AddReceiver* or *RemoveReceiver*.

A background thread is created for delivering events for each *IEventReceiver* object that is added to an *IEventSource* object. Only one thread is created per *IEventReceiver* object added to an *IEventSource* object, even if the *IEventReceiver* object is added more than once in order to subscribe to different events. If an *IEventReceiver* object is added to multiple *IEventSource* objects, one thread is still created for each *IEventSource* object.

Events sent from a specific *IEventSource* object to a specific *IEventReceiver* are always delivered sequentially on a single thread. The *IEventReceiver* will not receive additional events from that specific *IEventSource* until it has returned from *OnEvent*. Note that events sent to different *IEventReceiver* objects or received from different *IEventSource* objects are delivered on different threads and can be received simultaneously.

Members

`OnEvent`	This method is called by an *IEventSource* object when delivering an event in the form of an *IEventObject* object.

3.27.1 OnEvent

OnEvent is called by an *IEventSource* object when delivering an event in the form of an *IEventObject* object.

Syntax

.Net

C#	`void OnEvent( IEventObject  eventObj, System.IntPtr  pContext )`
C++	`void OnEvent( IEventObject^ eventObj, System::IntPtr pContext )`
VB	`Sub OnEvent( eventObj As IEventObject, pContext As System.IntPtr )`

ObjC

```
- (void) onEvent: (P1CaptureCore_EventObject *) eventObj
                  context: (void *) pContext
```

Parameters

eventObj — An *IEventObject* object describing an event received from an *IEventSource* object.

pContext — An optional pointer argument defined when the *IEventReceiver* was subscribed to events by calling the *IEventSource* method *AddReceiver*.

Remarks

It is generally a good idea to handle events as quickly as possible, so that other events are not overly delayed in being delivered. Tasks that take a lengthy time should be performed on another thread, instead of in *OnEvent*.

Calls to *OnEvent* are made in a background thread. There is a single background thread per *IEventSource* object that this *IEventReceiver* is added to. Events from a specific *IEventSource* object are delivered sequentially on this thread. However, if the *IEventReceiver* is added to multiple *IEventSource* objects, events are delivered on different threads and possibly at the same time. An implementation of *OnEvent* should take care to use thread-synchronization mechanisms if added to more than one *IEventSource* object.

If the *IEventSource* method *RemoveReceiver* is called for this *IEventReceiver* at the same time that *OnEvent* is called by the background thread, the call to *RemoveReceiver* will wait until *OnEvent* is completed. It is important that *OnEvent* does not wait on any threads, or make calls that wait on any threads, that could directly or indirectly call *RemoveReceiver* for this *IEventReceiver* object. Otherwise, a deadlock will occur. It is, however, completely safe to call *RemoveReceiver* for this or other *IEventReceiver* objects directly from *OnEvent*.

3.28 IEventObject (P1CaptureCore_EventObject)

The *IEventObject* class contains information about an event received from an *IEventSource* derived class. *IEventObject* objects are read-only objects and are described by a minimum of an event ID. When an event is sent by an *IEventSource* object, an *IEventObject* is delivered to each *IEventReceiver* added to the *IEventSource* and subscribed to the event's event ID.

IEventObject objects can have an optional number of value arguments (*IEventArgument*). This number of arguments and their definition depends upon the individual event. It can include for example the property ID of a changed property value. See the specific *IEventSource* derived class for a list of possible event IDs and their arguments.

IEventObject is a child object of *IEventSource*, and inherits from *IChildObject*.

Members

`Id`	Returns the event ID of the *IEventObject*.
`NumberOfArguments`	Returns the number of optional event arguments.
`Argument`	Returns a specified event argument (*IEventArgument*).
Inherited from IChildObject	
`Parent`	Returns the parent *IEventSource* object of this object.

3.28.1 Id

Id returns the event ID of the *IEventObject*.

The event ID specifies which event the *IEventObject* represents. The event ID is set by the *IEventSource* object that posted the event. The event IDs for a specific class are described in the documentation for each class derived from *IEventSource*.

Syntax

.Net

C#	`uint Id { get; }`
C++	`property System::UInt32 Id` `{ System::UInt32 get(); }`
VB	`ReadOnly Property Id As UInteger`

ObjC

```
- (uint32_t) id
```

Return Value

A numerical value representing the ID of the *IEventObject*.

Remarks

The event ID of an *IEventObject* object doesn't change after the object is created.

3.28.2 NumberOfArguments

NumberOfArguments returns the number of optional event arguments.

Some events can include arguments which are retrieved via the *Argument* method. The number of available arguments is returned by this method.

Syntax

.Net

C#	`uint NumberOfArguments { get; }`
C++	`property System::UInt32 NumberOfArguments { System::UInt32 get(); }`
VB	`ReadOnly Property NumberOfArguments As UInteger`

ObjC

```
- (uint32_t) numberOfArguments
```

Return Value

The number of arguments that can be retrieved via the *Argument* method.

Remarks

The number of arguments in the *IEventObject* doesn't change after the object is created.

3.28.3 Argument

Argument returns a specified event argument (*IEventArgument*).

Syntax

.Net

C#	`IEventArgument  Argument( uint index )`
C++	`IEventArgument^ Argument( System::UInt32 index )`
VB	`Function Argument( index As UInteger ) As IEventArgument`

ObjC

```
- (P1CaptureCore_EventArgument *) argument: (uint32_t) index
```

Parameters

index	A zero based index specifying which argument to return. The first argument is zero, the last argument is one less than the number of arguments returned by *NumberOfArguments*.

Return Value

A reference to an *IEventArgument* object that corresponds to the *index* parameter. A NULL reference is returned if there is no argument for the specified *index*.

Remarks

An *IEventArgument* object is derived from the *IValueRead* class and represents simple values that can be passed with the event. Typical values are the ID of the *ICaptureProvider*, *ICamera*, *ICaptureImage*, *IProperty*, or *ICapability* object associated with the event. See the documentation for an event ID for the possible event arguments.

3.29 IEventArgument (P1CaptureCore_EventArgument)

The *IEventArgument* class describes an optional argument value for an instance of an *IEventObject* object. An *IEventArgument* object is a read-only object and inherits from *IValueRead*.

IEventObject objects can have an optional number of *IEventArgument* objects for a given event. For example, a possible event argument can be the property ID of a changed property value. Although *IEventArgument* objects are retrieved from a specific *IEventObject* object, *IEventArgument* objects are not child objects of *IEventObject*. They are part of the *IEventObject*.

Members

Inherited from IValueRead	
`ValueType`	Returns the value type (Boolean, integer, string, etc) of the object.
`IsUndefined`	Returns true if the object's value is undefined.
`GetValue`	Gets the value of the object if the type of the object and the type passed to *GetValue* are compatible. One can always get a string representation for all value types. *GetValue* is available on platforms that support overloading.
`GetValueBool`	Returns the value of the object if its value type is a Boolean.
`GetValueInt32`	Returns the value of the object if its value type is a 32-bit signed integer (or enumeration).
`GetValueUInt32`	Returns the value of the object if its value type is a 32-bit unsigned integer (or enumeration).
`GetValueInt64`	Returns the value of the object if its value type is a 64-bit signed integer.
`GetValueUInt64`	Returns the value of the object if its value type is a 64-bit unsigned integer.
`GetValueFloat64`	Returns the value of the object if its value type is a 64-bit floating point.
`GetValueString`	Returns the value of the object if its value type is a string, or a string representation of the value for all other value types.
`GetValueEnum`	Returns the value of the object if its value type is an enumeration (32-bit signed integer).
`GetValuePoint`	Returns the value of the object if its value type is a point (32-bit signed integer).
`GetValuePointFloat`	Returns the value of the object if its value type is a point (64-bit floating point).
`GetValueArea`	Returns the value of the object if its value type is an area (32-bit signed integer).
`GetValueAreaFloat`	Returns the value of the object if its value type is an area (64-bit floating point).

`GetValueRect`	Returns the value of the object if its value type is a rectangle (32-bit signed integer).
`GetValueRectFloat`	Returns the value of the object if its value type is a rectangle (64-bit floating point).
`Compare`	Compares this object's value to another object of the same value type, returning a signed integer representing if this object is less than, greater than or equal to the other object.

3.30 IProgressSource (P1CaptureCore_ProgressSource)

The *IProgressSource* base class provides a common set of functionality for classes that can report progress status for progress tasks. It does not exist as an object on its own, and is only accessible via a derived class.

IProgressSource is a parent to *IProgressStatus* objects.

Members

`GetProgress`	Returns the next *IProgressStatus* object in the progress queue for this object.

Events

General `(EnumGeneralEventId)`	
`kEventId_ProgressUpdate`	Indicates that a new *IProgressStatus* object has been queued by this object.

3.30.1 GetProgress

GetProgress returns the next *IProgressStatus* object in the progress queue for this object.

Syntax

.Net

C#	`IProgressStatus  GetProgress()`
C++	`IProgressStatus^ GetProgress()`
VB	`Function GetProgress As IProgressStatus`

ObjC

```
- (P1CaptureCore_ProgressStatus *) getProgress
```

Return Value

A reference to the next *IProgressStatus* object in the progress queue of the *IProgressSource* object. A NULL reference is returned if there are no *IProgressStatus* objects in the progress queue.

Remarks

GetProgress removes the *IProgressStatus* object from the progress queue. So subsequent calls to *GetProgress* will return new objects, or a NULL reference if there are no queued objects when the method is called.

3.30.2 kEventId_ProgressUpdate

This event is posted by the *IProgressSource* object when an *IProgressStatus* object has been added to the progress queue of the *IProgressSource* object.

Arguments

None

3.31 IProgressStatus (P1CaptureCore_ProgressStatus)

The *IProgressStatus* class provides progress status for an instance of a progress task for an *IProgressSource* class. It is essentially a read-only data structure, with an optional capability to cancel the progress task it is associated with.

IProgressStatus objects are created and queued by the *IProgressSource* class and retrieved by a call to *GetProgress*. When the object is queued, a *kEventId_ProgressUpdate* event is posted by the *IProgressSource* class. Since there is a lag between the creation of the object and its retrieval by the application, the status reported may not represent the current status, but a snapshot of the status in the recent past. Therefore, an application should try to retrieve *IProgressStatus* objects as quickly as possible.

IProgressStatus is a child object of *IProgressSource,* and inherits from *IChildObject*.

Members

`Id`	Returns the progress status id for the *IProgressStatus* object, identifying the type of progress task the object represents.
`Instance`	Returns an unique number that is the same for all *IProgressStatus* objects that represent the same instance of a progress task.
`Description`	Returns an optional string describing the progress task.
`Detail`	Returns an optional string with additional detail of what the progress task is currently performing.
`Unit`	Returns the unit of the values returned by *Current* and *End* as an optional string. For example “MB” for megabytes.
`Current`	Returns a numerical value representing the current progress state of the progress task. The value is always <= *End*.
`End`	Returns a numerical value representing the end progress state of the progress task. This could be zero, if no such value is known.
`Percent`	Returns the current progress state as a percent, if *End* is defined (not zero).
`ElapsedTime`	Returns the time in milliseconds since the progress task started.
`IsDone`	Returns true if the progress task is complete. If true, this is the last *IProgressStatus* object for the progress task instance.
`IsCancelled`	Returns true if the progress task has been cancelled. If true, this is the last *IProgressStatus* object for the progress task instance.
`CanCancel`	Returns true if the progress task can be cancelled by calling *Cancel*.
`Cancel`	Cancels the represented progress task, if it is still active. *CanCancel* must be true to call this method.
Inherited from IChildObject	
`Parent`	Returns the parent *IEventSource* object of this object.

3.31.1 Id

Id returns the progress status id for the *IProgressStatus* object, identifying the type of progress task the object represents.

Syntax

.Net

C#	`uint Id { get; }`
C++	`property System::UInt32 Id` `{ System::UInt32 get(); }`
VB	`ReadOnly Property Id As UInteger`

ObjC

```
- (uint32_t) id
```

Return Value

The progress status id for the *IProgressStatus* object. See the description for each specific *IProgressSource* derived class for possible progress status id values.

Remarks

The ID of an *IProgressStatus* object doesn't change after the object is created. It is also the same for all *IProgressStatus* objects of the same task instance.

3.31.2 Instance

Instance returns an unique number that is the same for all *IProgressStatus* objects that represent the same instance of a progress task.

Syntax

.Net

C#	`uint Instance { get; }`
C++	`property System::UInt32 Instance` `{ System::UInt32 get(); }`
VB	`ReadOnly Property Instance As UInteger`

ObjC

```
- (uint32_t) instance
```

Return Value

An unique number representing the progress task instance that created the *IProgressStatus* object.

Remarks

All instances of a progress task have an unique instance number, which is assigned to all *IProgressStatus* objects created by that task. This number can be used to distinguish between the progress status of different tasks that are running at the same time, such as capturing two images at the same time from an *ICamera* object.

The instance number of an *IProgressStatus* object doesn't change after the object is created.

3.31.3 Description

Description returns an optional string describing the progress task.

Syntax

.Net

C#	`string Description { get; }`
C++	`property System::String^ Description` `{ System::String^ get(); }`

VB	`ReadOnly Property Description As String`

ObjC

```
- (NSString *) description
```

Return Value

An optional string describing the progress task the status object describes. The string is generally short and useful for providing a title or heading for a progress task. A NULL reference or an empty string is returned if no description is available.

Remarks

The description string of an *IProgressStatus* object doesn't change after the object is created. Its value is usually the same in all *IProgressStatus* objects of the same task instance.

3.31.4 Detail

Detail returns an optional string with additional detail of what the progress task is currently performing. This is generally useful when the progress task is divided into subtasks, or has multiple objects to perform the task upon, such as saving multiple files. The string can be the name of the subtask, or the name of the object, such as a file name.

Syntax

.Net

C#	`string Detail { get; }`
C++	`property System::String^ Detail` `{ System::String^ get(); }`
VB	`ReadOnly Property Detail As String`

ObjC

```
- (NSString *) detail
```

Return Value

An optional string with detail of what the progress task is currently doing. A NULL reference or an empty string is returned if no detail is available.

Remarks

The detail string of an *IProgressStatus* object doesn't change after the object is created. Its value is different in subsequent *IProgressStatus* objects of the same task instance.

3.31.5 Unit

Unit returns the unit of the values returned by *Current* and *End* as an optional string. For example "MB" for megabytes.

Syntax

.Net

C#	`string Unit { get; }`
C++	`property System::String^ Unit` `{ System::String^ get(); }`
VB	`ReadOnly Property Unit As String`

ObjC

```
- (NSString *) unit
```

Return Value

An optional string describing the unit of the values returned by *Current* and *End*. A NULL reference or an empty string is returned if the unit is not defined.

Remarks

The unit string of an *IProgressStatus* object doesn't change after the object is created. Its value is usually the same in all *IProgressStatus* objects of the same task instance.

3.31.6 Current

Current returns a numerical value representing the current progress state of the progress task.

Syntax

.Net

C#	`ulong Current { get; }`
C++	`property System::UInt64 Current` `{ System::UInt64 get(); }`
VB	`ReadOnly Property Current As ULong`

ObjC

```
- (uint64_t) current
```

Return Value

A numerical value representing the current progress state of the progress task. The value is always <= *End*. It returns zero if *End* is not defined (zero).

Remarks

The value of the current progress state of an *IProgressStatus* object doesn't change after the object is created. Its value is different in subsequent *IProgressStatus* objects of the same task instance.

3.31.7 End

End returns a numerical value representing the end progress state of the progress task.

Syntax

.Net

C#	`ulong End { get; }`
C++	`property System::UInt64 End` `{ System::UInt64 get(); }`
VB	`ReadOnly Property End As ULong`

ObjC

```
- (uint64_t) end
```

Return Value

A numerical value representing the end length or state of the progress task. It will return zero for tasks without a defined length, such as waiting for a device to respond to a request.

Remarks

If the progress task has an undefined length, an application can still use the value returned by *ElapsedTime* to provide the user with a sense of progress.

The value of the end progress state of an *IProgressStatus* object doesn't change after the object is created. Its value is usually the same in all *IProgressStatus* objects of the same task instance, but is allowed to change.

3.31.8 Percent

Percent returns the current progress state as a percent, if *End* is defined (not zero).

Syntax

.Net

C#	`double Percent { get; }`
C++	`property double Percent` `{ double get(); }`
VB	`ReadOnly Property Percent As Double`

ObjC

```
- (double) percent
```

Return Value

A floating point value representing the current progress state as a percent. This value is equivalent to dividing *Current* by *End*. It returns 0.0 if *End* is undefined (zero).

Remarks

The percent value of an *IProgressStatus* object doesn't change after the object is created. Its value is different in subsequent *IProgressStatus* objects of the same task instance.

3.31.9 Elapsed Time

Returns the time in milliseconds since the progress task started.

Syntax

.Net

C#	`uint ElapsedTime { get; }`
C++	`property System::UInt32 ElapsedTime` `{ System::UInt32 get(); }`
VB	`ReadOnly Property ElapsedTime As UInteger`

ObjC

```
- (uint32_t) elapsedTime
```

Return Value

The time in milliseconds since the progress task started, at the moment the *IProgressStatus* object was created and queued. *ElapsedTime* doesn't include any time between the creation of the object and a call to the method.

Remarks

The percent value of an *IProgressStatus* object doesn't change after the object is created. Its value is different in subsequent *IProgressStatus* objects of the same task instance.

3.31.10 IsDone

IsDone returns true if the progress task is complete.

Syntax

.Net

C#	`bool IsDone()`
C++	`bool IsDone()`
VB	`Function IsDone As Boolean`

ObjC

```
- (BOOL) isDone
```

Return Value

A Boolean value that is true if the progress task is complete. If true, this is the last *IProgressStatus* object for the progress task instance.

3.31.11 IsCancelled

Returns true if the progress task has been cancelled.

Syntax

.Net

C#	`bool IsCancelled()`
C++	`bool IsCancelled()`
VB	`Function IsCancelled As Boolean`

ObjC

```
- (BOOL) isCancelled
```

Return Value

A Boolean value that is true if the progress task has been cancelled. If true, this is the last *IProgressStatus* object for the progress task instance.

Remarks

A progress task can be cancelled internally by an object, usually in response to an error, or it may be cancelled by calling the *IProgressStatus Cancel* method for the progress task instance that the application wishes to cancel.

3.31.12 CanCancel

Returns true if the progress task can be cancelled by calling *Cancel*.

Syntax

.Net

C#	`bool CanCancel()`
C++	`bool CanCancel()`
VB	`Function CanCancel As Boolean`

ObjC

```
- (BOOL) canCancel
```

Return Value

A Boolean value that is true if the progress task can be cancelled by calling *Cancel*.

Remarks

This method can be used to determine if a cancel button is displayed by the application for the progress task.

3.31.13 Cancel

Cancels the represented progress task, if it is still active.

Syntax

.Net

C#	`void Cancel()`
C++	`void Cancel()`
VB	`Sub Cancel`

ObjC

```
- (void) cancel
```

Remarks

Not all progress tasks can be cancelled. Calls to this method are ignored if *CanCancel* is false. In addition, calling cancel after the progress task is completed will have no affect.

4 Enumeration Reference

The general enumerations not used for capabilities, properties and events are described in this section.

4.1 EnumErrorType

EnumErrorType defines some of the possible error types that an *IErrorObject* can represent. Not all error types are defined, just those commonly encountered.

kErrorType_Unknown	An unknown error type.
kErrorType_System	An operating system error.
kErrorType_StdErrno	A standard C errno error.
kErrorType_StdException	A standard C++ std::exception error.
kErrorType_CaptureCore	CaptureCore errors. See Error Reference.

4.2 EnumValueType

EnumValueType defines the data type of the value represented by an *IValueRead* or *IValueWrite* derived class.

kValueType_Undefined	An undefined or unknown value type.
kValueType_Bool	A Boolean value that can be true or false.
kValueType_Int32	A signed 32-bit integer value.
kValueType_UInt32	An unsigned 32-bit integer value.
kValueType_Int64	A signed 64-bit integer value.
kValueType_UInt64	An unsigned 64-bit integer value.
kValueType_Float64	A 64-bit floating point value.
kValueType_String	A Unicode string value.
kValueType_Enum	An enumeration value compatible with a signed/unsigned 32-bit integer.
kValueType_Point	A coordinate point (x, y) specified by two signed 32-bit integer values.
kValueType_PointFloat	A coordinate point (x, y) specified by two 64-bit floating point values.
kValueType_Area	An area size (width, height) specified by two signed 32-bit integer values.
kValueType_AreaFloat	An area size (width, height) specified by two 64-bit floating point values.
kValueType_Rect	A rectangle (x, y, width, height) specified by four signed 32-bit integer values.
kValueType_RectFloat	A rectangle (x, y, width, height) specified by four floating point values.

4.3 EnumListAccess

EnumListAccess defines the list access rights that a caller has for an object of a list class derived from *IObjectList*. *EnumListAccess* values can be combined together in a bitmask.

kListAccess_View	The list object may be iterated and items may be retrieved for viewing.
kListAccess_Modify	The list object may be iterated and items may be retrieved for modifying.
kListAccess_Insert	New items may be added to the list object.
kListAccess_Remove	Items may be removed from the list object.
kListAccess_All	All the above access rights.

4.4 EnumCaptureCoreName

EnumCaptureCoreName defines the name strings that can be returned by some CaptureCore *Name* methods.

`kCaptureCoreName_Vendor`	A vendor specific name string, provided by the manufacturer associated with the object.
`kCaptureCoreName_Long`	A name string provided by Phase One.
`kCaptureCoreName_Short`	A short name string, provided by Phase One, that is guaranteed to be 20 characters or less.

4.5 EnumImageType

EnumImageType defines the image types that an image object, such as *IImageData*, can be.

`kImageType_Undefined`	An undefined or unknown image type.
`kImageType_Pixel`	An uncompressed pixel image.
`kImageType_Jpeg`	A JPEG image.

4.6 EnumColorType

EnumColorType defines possible color types that the pixels of an image object, such as *IImageData*, can be. The order of color channels in the name of the enumeration match the order of the color channels in the pixel.

`kColorType_Undefined`	An undefined or unknown color type.
`kColorType_RGB_8`	Red, green, blue. 8-bits per channel (24-bit pixel).
`kColorType_BGR_8`	Blue, green, red. 8-bits per channel (24-bit pixel).
`kColorType_RGBA_8`	Red, green, blue, alpha. 8-bits per channel (32-bit pixel).
`kColorType_BGRA_8`	Blue, green, red, alpha. 8-bits per channel (32-bit pixel).
`kColorType_ARGB_8`	Alpha, red, green, blue. 8-bits per channel (32-bit pixel).
`kColorType_ABGR_8`	Alpha, blue, green, red. 8-bits per channel (32-bit pixel).
`kColorType_RGB_16`	Red, green, blue. 16-bits per channel (48-bit pixel).
`kColorType_BGR_16`	Blue, green, red. 16-bits per channel (48-bit pixel).

4.7 EnumImageOrientation

EnumImageOrientation defines the orientations that an image object, such as *IImageData*, can have.

`kImageOrientation_0`	The image capture device was rotated 0 degrees (i.e. not rotated).
`kImageOrientation_90`	The image capture device was rotated 90 degrees clockwise.
`kImageOrientation_180`	The image capture device was rotated 180 degrees.
`kImageOrientation_270`	The image capture device was rotated 270 degrees clockwise.
`kImageOrientation_TopLeft`	The first pixel row/column is the top/left edge of the image. The same as `kImageOrientation_0`.
`kImageOrientation_RightTop`	The first pixel row/column is the right/top edge of the image. The same as `kImageOrientation_90`.
`kImageOrientation_BottomRight`	The first pixel row/column is the bottom/right edge of the image. The same as `kImageOrientation_180`.
`kImageOrientation_LeftBottom`	The first pixel row/column is the left/bottom edge of the image. The same as `kImageOrientation_270`.

4.8 EnumCameraType

EnumCameraType defines the camera types that an *ICamera* object can have, and that can be retrieved by the *kCameraProperty_Type* property.

`kCameraType_DB`	A digital back.
`kCameraType_DSLR`	A DSLR (digital single lens reflex) camera.

4.9 EnumCameraOrientationMode

EnumCameraOrientationMode defines the possible orientation modes that an *ICamera* object can be set to via the *kCameraProperty_CameraOrientationMode*. The mode used during the capture of an image is stored in *kCaptureImageProperty_CameraOrientationMode*.

`kCameraOrientationMode_Undefined`	An undefined or unknown camera orientation mode.
`kCameraOrientationMode_Auto`	Use the internal orientation sensor to determine orientation.
`kCameraOrientationMode_0`	Set the orientation to 0 degrees. Do not use the internal orientation sensor.
`kCameraOrientationMode_90`	Set the orientation to 90 degrees clockwise. Do not use the internal orientation sensor.
`kCameraOrientationMode_180`	Set the orientation to 180 degrees. Do not use the internal orientation sensor.
`kCameraOrientationMode_270`	Set the orientation to 270 degrees clockwise. Do not use the internal orientation sensor.

5 Error Reference

CaptureCore returns errors via the *IErrorObject* interface. CaptureCore can return different types of errors, depending upon the source of the error. Some errors originate in the operating system, some within a development framework, some from device drivers, and some from CaptureCore itself. For each type of error, there are many possible errors, each with their own unique error number for that type. Error numbers are not unique across different error types.

It is beyond the scope of this document to describe all the errors for error types originating outside of CaptureCore. Generally, the *IErrorObject* provides enough description strings to display the error to the user. However, in some situations it may be of use to the application to test for a specific CaptureCore error. The following table lists the error enumerations for CaptureCore errors. CaptureCore errors have the error type *kErrorType_CaptureCore* (see *EnumErrorType*).

5.1 CaptureCore Errors

General `(EnumCaptureCoreError)`	
`kCaptureCoreError_InvalidParameter`	The parameter is incorrect or not supported.
`kCaptureCoreError_InvalidType`	The data type is incorrect or not supported.
`kCaptureCoreError_InvalidData`	The data is invalid.
`kCaptureCoreError_OutOfRange`	The value is out of range.
`kCaptureCoreError_InvalidSize`	The size or length is incorrect.
`kCaptureCoreError_InvalidIndex`	The index or identifier is incorrect.
`kCaptureCoreError_InvalidSyntax`	The syntax is incorrect.
`kCaptureCoreError_NotImplemented`	The functionality is not implemented.
`kCaptureCoreError_InvalidRequest`	The request is invalid.
`kCaptureCoreError_InvalidState`	The current state is incorrect.
`kCaptureCoreError_NotSupported`	The resource or request is not supported.
`kCaptureCoreError_NotAvailable`	The resource or request is currently not available.
`kCaptureCoreError_NotInitialized`	The resource is not initialized.
`kCaptureCoreError_AlreadyInitialized`	The resource is already initialized.
`kCaptureCoreError_NotOpen`	The resource is not open.
`kCaptureCoreError_AlreadyOpen`	The resource is already opened.

kCaptureCoreError_AccessDenied	Access is denied. The resource or request is not available.
kCaptureCoreError_AccessDeniedWrite	Access is denied. The value or resource cannot be set or written to.
kCaptureCoreError_AccessDeniedRead	Access is denied. The value or resource cannot be retrieved or read from.
kCaptureCoreError_NotConnected	The resource is not connected.
kCaptureCoreError_NotEnoughMemory	Not enough memory is available.
kCaptureCoreError_UnexpectedError	An unexpected error occurred.
kCaptureCoreError_UnexpectedException	An unexpected exception occurred.
kCaptureCoreError_UnexpectedResult	An unexpected result occurred.
kCaptureCoreError_LimitExceeded	A limit is exceeded.
kCaptureCoreError_NotFound	The name, item, or resource is not found.
kCaptureCoreError_Timeout	The request did not complete within the specified timeout period.
kCaptureCoreError_UnspecifiedError	Unspecified error.
kCaptureCoreError_CameraNotConnected	The camera is not connected.
Phase One device specific (EnumPhaseOneCaptureCoreError)	
kP1CaptureCoreError_HostStorageMode	The camera is not configured for IEEE 1394 storage.
kP1CaptureCoreError_MacCreateLocalIsochPortError	Mac OS only. Could not create local isochronous port. There is insufficient memory below the 2GB memory boundary for the operating system to setup an isochronous FireWire transfer port between the host and the device.

6 Capability Reference

The following tables list the defined capabilities for each class that supports them. The tables list the typical value type for each capability, however, capabilities are not required to be of this value type. An application should be prepared to handle any value type for each capability, or at the very least ignore gracefully a capability with a value type different than expected. A string value can always be retrieved for each capability, regardless of the actual value type.

6.1 ICamera (P1CaptureCore_Camera)

Capability	**Typical Value Type**	**Description**
General `(EnumCameraCapabilityId)`		
`kCameraCapability_Capture`	Bool	If true, general capturing functionality is supported, such as *StartCapture*, *StopCapture*, *PauseCapture*, *GetNextCaptureImage*, *GetCaptureImageQueue*, and *MaxCaptureQueueSize* methods.
`kCameraCapability_PauseCapture`	Bool	If true, the *PauseCapture* method is supported.
`kCameraCapability_PauseCaptureAndTransfer`	Bool	If true, the *bPauseTransfer* parameter of *PauseCapture* method is supported, allowing it to pause the transfer of images in addition to pausing their capture.
`kCameraCapability_WaitOnPending`	Bool	If true, the *bWaitOnPending* parameter of the *StopCapture* method is supported, allowing it to optionally wait on pending images before stopping capture.
`kCameraCapability_PendingImageCount`	Bool	If true, the *PendingImageCount* method is supported.
`kCameraCapability_ShutterRelease`	Bool	If true, the *ShutterRelease* method is supported.
`kCameraCapability_MaxCaptureQueueSize`	Bool	If true, the maximum capture queue size can be set via the *MaxCaptureQueueSize* methods.
`kCameraCapability_LiveView`	Bool	If true, this device supports Live View functionality.
Phase One device specific `(EnumPhaseOneCameraCapabilityId)`		
`kP1CameraCapability_ColorRGB`	Bool	If true, this is a RGB color device.
`kP1CameraCapability_ColorBW`	Bool	If true, this is a monochrome color device.

6.2 ICaptureImage (P1CaptureCore_CaptureImage)

Capability	Typical Value Type	Description
General `(EnumCaptureImageCapabilityId)`		
`kCaptureImageCapability_Thumbnail`	Bool	If true, the *GetThumbnail* method is supported.

7 Property Reference

The following tables list the defined properties for each class that supports them. The tables list the typical value type for each property, however, properties are not required to be of this value type. An application should be prepared to handle any value type for each property, or at the very least ignore gracefully a property with a value type different than expected. A string value can always be retrieved for each property, regardless of the actual value type.

The tables also list the typical access an application has to each property, read-only or read/write. A property may be read-only on one device, read/write on another, or not even present. The application should be prepared to handle missing properties or properties with a different access than expected.

7.1 ICaptureProvider (P1CaptureCore_CaptureProvider)

Property	Typical Value Type	Typical Access	Description
General `(EnumCaptureProviderPropertyId)`			
`kCaptureProviderProperty_ManufacturerName`	String	Read	Manufacturer's name.

7.2 ICamera (P1CaptureCore_Camera)

Property	Typical Value Type	Typical Access	Description
General (EnumCameraPropertyId)			
kCameraProperty_ManufacturerName	String	Read	Manufacturer's name.
kCameraProperty_Model	String/Enum	Read	Device model.
kCameraProperty_SerialNumber	String/Number	Read	Serial number.
kCameraProperty_FirmwareVersion	String/Number	Read	Firmware version.
kCameraProperty_Description	String	Read	Description of the device.
kCameraProperty_Type	Enum	Read	The type of camera device. See *EnumCameraType*.
kCameraProperty_MaxTransferSpeed	UInt64	Read	Maximum transfer speed in bytes per second.
kCameraProperty_NumberOfImagesTaken	UInt32	Read	Current number of images taken by the device.
kCameraProperty_HostMaxCaptureQueueSize	UInt32	Read/Write	The maximum number of images to queue on the host. Same as *MaxCaptureQueueSize* methods.
kCameraProperty_HostStorageCapacity	UInt64	Read/Write	Number of available bytes on the host for storing images.
kCameraProperty_ImageSize	String/Enum	Read/Write	The size of an image (e.g. large, medium, small). Related to resolution.
kCameraProperty_ImageArea	String/Enum	Read/Write	The physical area on the sensor to acquire images with.
kCameraProperty_WhiteBalanceMode	String/Enum	Read/Write	The white balance mode to use during capture (e.g. Auto, Flash, Daylight, etc.)
kCameraProperty_FileFormat	Enum	Read/Write	The file format of captured images.
kCameraProperty_ImageCompression	Enum	Read/Write	Image compression setting (e.g. IIQ L or IIQ S).
kCameraProperty_ImageMaximumSize	UInt32	Read	Maximum size in bytes of an image for this device (worst case).
kCameraProperty_ImageTypicalSize	UInt32	Read	Typical size in bytes of an image for the current settings.
kCameraProperty_ThumbnailMaxDimension	UInt32	Read/Write	The default maximum dimension in pixels of generated and embedded thumbnail images in captured images.
kCameraProperty_ExposureISO	UInt32/Enum	Read/Write	Exposure ISO (e.g. ISO 100).
kCameraProperty_ShutterSpeed	Float64/Enum	Read/Write	Shutter speed in seconds (e.g. 1.4 s or 1/125 s).
kCameraProperty_Aperture	Float64/Enum	Read/Write	Aperture value in f-stops (e.g. f/22).
kCameraProperty_ExposureBias	Float64/Enum	Read/Write	Exposure bias in exposure steps (e.g. -1.5 or 3.0).
kCameraProperty_ExposureMode	Enum	Read/Write	Exposure mode (e.g. Auto, Manual, Auto bracket).

Property	Typical Value Type	Typical Access	Description
kCameraProperty_ExposureStep	Enum	Read/Write	Exposure step setting for *kCameraProperty_ExposureProgram*, *kCameraProperty_Aperture*, and *kCameraProperty_ExposureBias* properties (e.g. 1, 1/2, or 1/3).
kCameraProperty_ExposureProgram	Enum	Read/Write	Exposure program (e.g. P, Av, Tv or M).
kCameraProperty_CameraOrientationMode	Enum	Read/Write	Allows specification of the camera orientation (Auto, 0, 90, 180, 270). In Auto mode (the default), the camera orientation is determined by a rotation sensor in the device. The image orientation is determined by both the source orientation and the camera orientation. See *EnumCameraOrientationMode*.
Phase One device specific (EnumPhaseOneCameraEventId)			
kP1CameraProperty_MainCodeVersion	String	Read	Main code (firmware) version.
kP1CameraProperty_BootCodeVersion	String	Read	Boot code version.
kP1CameraProperty_FPGACodeVersion	String	Read	FPGA code version.
kP1CameraProperty_CPLDCodeVersion	String	Read	CPLD code version.
kP1CameraProperty_PAVRCodeVersion	String	Read	PAVR code version.
kP1CameraProperty_UAVRCodeVersion	String	Read	UAVR code version.
kP1CameraProperty_TGENCodeVersion	String	Read	TGEN code version.
kP1CameraProperty_HardwareConfig	UInt32	Read	Hardware configuration value.
kP1CameraProperty_SensorType	Enum	Read	Sensor type.
kP1CameraProperty_SensorBaseISO	UInt32	Read	Lowest ISO value.
kP1CameraProperty_SensorTemperature	Float64	Read	The current sensor temperature in degrees Celsius.
kP1CameraProperty_SensorWidth	UInt32	Read	Width of the sensor in pixels.
kP1CameraProperty_SensorHeight	UInt32	Read	Height of the sensor in pixels.
kP1CameraProperty_SensorActiveWidth	UInt32	Read	Width of the active area on the sensor in pixels. The active area is the area on the sensor where it is exposed.
kP1CameraProperty_SensorActiveHeight	UInt32	Read	Height of the active area on the sensor.
kP1CameraProperty_SensorActiveXOffset	UInt32	Read	X-offset of the active area on the sensor.
kP1CameraProperty_SensorActiveYOffset	UInt32	Read	Y-offset of the active area on the sensor.
kP1CameraProperty_SensorOrientation	Int32	Read	Orientation of the sensor relative to the device in degrees (0, 90, 180, 270). The angle is measured clockwise (to the right) from vertical.

Property	Typical Value Type	Typical Access	Description
kP1CameraProperty_SourceWidth	UInt32	Read	Width of the image source in pixels. The image source represents the effective capture area of the sensor in the current device mode. For example, if sub-sampling is enabled, the image source is reduced compared to the sensor.
kP1CameraProperty_SourceHeight	UInt32	Read	Height of the image source in pixels.
kP1CameraProperty_SourceActiveWidth	UInt32	Read	Width of the active area on the image source in pixels. The active area is the area on the image source where it is exposed.
kP1CameraProperty_SourceActiveHeight	UInt32	Read	Height of the active area on the image source.
kP1CameraProperty_SourceActiveXOffset	UInt32	Read	X-offset of the active area on the image source.
kP1CameraProperty_SourceActiveYOffset	UInt32	Read	Y-offset of the active area on the image source.
kP1CameraProperty_SourceOrientation	Int32	Read	Orientation of the image source relative to the device in degrees (0, 90, 180, 270). The angle is measured clockwise (to the right) from vertical.
kP1CameraProperty_WhiteBalanceRed	Float64	Read/Write	White balance red value (0.0 to 1.0).
kP1CameraProperty_WhiteBalanceGreen	Float64	Read/Write	White balance green value (0.0 to 1.0).
kP1CameraProperty_WhiteBalanceBlue	Float64	Read/Write	White balance blue value (0.0 to 1.0). The white balance is only updated on the device after writing the blue value.
kP1CameraProperty_CameraOrientationMode	Enum	Read/Write	Obsolete. Use *kCameraProperty_CameraOrientationMode* instead.
kP1CameraProperty_SensorPlus	Enum	Read/Write	Sensor+ mode.
kP1CameraProperty_DisableBlackUpdate	Bool	Read/Write	Disables updating of the black calibration. To improve image quality, a black calibration image is sometimes captured following a normal capture. If this property is set to true, the camera will not update the black calibration, which will increase the sustained capture rate, but there is a risk of reduced image quality.
kP1CameraProperty_UseRemoteCaptureSettings	Bool	Read/Write	Controls whether the values of the camera control properties (*kCameraProperty_ExposureProgram*, *kCameraProperty_Aperture*, *kCameraProperty_ExposureBias*, and *kCameraProperty_ExposureStep*) or the current settings on the camera are used for remote captures. Remote captures are those made via a *ShutterRelease* call and not via the camera. Set to true to use the camera control properties.
kP1CameraProperty_SafeMirrorUp	Bool	Read/Write	If true, enables safe mirror up mode, which delays some of the internal timing of the "mirror up" functionality, and is necessary for proper functioning in some situations. Primarily for H 20 models.
kP1CameraProperty_PowerMode	Enum	Read/Write	Power mode setting of the device (e.g. normal, low or ultra low).

Property	Typical Value Type	Typical Access	Description
`kP1CameraProperty_ShutterLatency`	Enum	Read/Write	Shutter latency setting of the device (e.g. long/normal or short/zero).

7.3 ICaptureImage (P1CaptureCore_CaptureImage)

Property	Typical Value Type	Typical Access	Description
General (EnumCapturePropertyId)			
kCaptureImageProperty_ManufacturerName	String	Read	Manufacturer's name.
kCaptureImageProperty_Model	String/Enum	Read	Device model.
kCaptureImageProperty_SerialNumber	String/Number	Read	Serial number.
kCaptureImageProperty_FirmwareVersion	String/Number	Read	Firmware version.
kCaptureImageProperty_DeviceDescription	String	Read	Description of the device.
kCaptureImageProperty_SoftwareDescription	String	Read	Description of the software used to capture the image.
kCaptureImageProperty_PlatformDescription	String	Read	Description of the host platform the image was captured on.
kCaptureImageProperty_Width	UInt32	Read	Width of the image in pixels.
kCaptureImageProperty_Height	UInt32	Read	Height of the image in pixels.
kCaptureImageProperty_ActiveWidth	UInt32	Read	Width of the active area of the image in pixels. The active area is the area within the image where the image was exposed.
kCaptureImageProperty_ActiveHeight	UInt32	Read	Height of the active area of the image in pixels.
kCaptureImageProperty_ActiveXOffset	UInt32	Read	X-offset in pixels of the active area of the image.
kCaptureImageProperty_ActiveYOffset	UInt32	Read	Y-offset in pixels of the active area of the image.
kCaptureImageProperty_ImageOrientation	Int32	Read	Orientation of the image relative to the device in degrees (0, 90, 180, 270).
kCaptureImageProperty_FileFormat	Enum	Read	Image file format.
kCaptureImageProperty_ImageCompression	Enum	Read	Image compression setting (e.g. IIQ L or IIQ S).
kCaptureImageProperty_ImageSize	String/Enum	Read	Size of the image as a setting (e.g. large, medium, small).
kCaptureImageProperty_FileSize	UInt32	Read	Size of the image in bytes.
kCaptureImageProperty_ThumbnailMaxDimension	UInt32	Read	The maximum dimension of generated and embedded thumbnail images. Note: might be read-only. See also kCameraProperty_ThumbnailMaxDimension.
kCaptureImageProperty_ThumbnailWidth	UInt32	Read	The width in pixels of the embedded thumbnail image.
kCaptureImageProperty_ThumbnailHeight	UInt32	Read	The height in pixels of the embedded thumbnail image.
kCaptureImageProperty_ThumbnailSize	UInt32	Read	The size in bytes of the embedded thumbnail image.
kCaptureImageProperty_DefaultFilenameExtension	String	Read	Default file name extension (e.g. iiq or tif).

Property	Typical Value Type	Typical Access	Description
kCaptureImageProperty_ExposureISO	UInt32/Enum	Read	Exposure ISO (e.g. ISO 100).
kCaptureImageProperty_ShutterSpeed	Float64/Enum	Read	Shutter speed in seconds (e.g. 1.4 s or 1/125 s).
kCaptureImageProperty_ShutterSpeedApexValue	Float64	Read	Shutter speed as APEX value: Log2(1 / speed)
kCaptureImageProperty_Aperture	Float64/Enum	Read	Aperture value in f-stops (e.g. f/22).
kCaptureImageProperty_ApertureApexValue	Float64	Read	Aperture as APEX value: Log2(aperture^2)
kCaptureImageProperty_ExposureBias	Float64	Read	Exposure bias in exposure steps (e.g. -1.5 or 3.0).
kCaptureImageProperty_ExposureMode	Enum	Read	Exposure mode (e.g. Auto, Manual, Auto bracket).
kCaptureImageProperty_ExposureProgram	Enum	Read	Exposure program (e.g. P, Av, Tv or M).
kCaptureImageProperty_CameraOrientationMode	Enum	Read	The camera orientation mode at the time of capture (Auto, 0, 90, 180, 270). In Auto mode (the default), the camera orientation was determined by a rotation sensor in the device. The final image orientation is determined by both the source orientation and the camera orientation. See *EnumCameraOrientationMode*.
kCaptureImageProperty_FocalLength	Float64	Read	Focal length in mm.
kCaptureImageProperty_Timestamp	UInt64	Read	Timestamp for this image, in seconds since January 1st, 1970 0:00.
kCaptureImageProperty_CameraCaptureNumber	UInt32	Read	Capture number for this image set by the device.
kCaptureImageProperty_SoftwareCaptureNumber	UInt32	Read	CaptureCore generated capture number for this image.
kCaptureImageProperty_CameraAngle	Float64	Read	Device angle in degrees at the time the image was captured. The angle is measured clockwise (to the right) from vertical. For example, rotating the device to left will result in a camera angle of 270.0 or -90.0 degrees.
Phase One device specific (EnumPhaseOneCameraEventId)			
kP1CaptureImageProperty_HardwareConfig	UInt32	Read	A device specific hardware configuration value.
kP1CaptureImageProperty_SensorType	Enum	Read	Sensor type.
kP1CaptureImageProperty_SensorBaseISO	UInt32	Read	Lowest ISO value.
kP1CaptureImageProperty_SensorTemperature	Float64	Read	The sensor temperature during image capture in degrees Celsius.
kP1CaptureImageProperty_SensorWidth	UInt32	Read	Width of the sensor in pixels.
kP1CaptureImageProperty_SensorHeight	UInt32	Read	Height of the sensor in pixels.
kP1CaptureImageProperty_SensorActiveWidth	UInt32	Read	Width of the active area on the sensor in pixels. The active area is the area on the sensor where it is exposed.
kP1CaptureImageProperty_SensorActiveHeight	UInt32	Read	Height of the active area on the sensor.

Property	Typical Value Type	Typical Access	Description
kP1CaptureImageProperty_SensorActiveXOffset	UInt32	Read	X-offset of the active area on the sensor.
kP1CaptureImageProperty_SensorActiveYOffset	UInt32	Read	Y-offset of the active area on the sensor.
kP1CaptureImageProperty_SensorOrientation	Int32	Read	Orientation of the sensor relative to the device in degrees (0, 90, 180, 270). The angle is measured clockwise (to the right) from vertical.
kP1CaptureImageProperty_SourceWidth	UInt32	Read	Width of the image source in pixels. The image source represents the effective capture area of the sensor in the current device mode. For example, if sub-sampling is enabled, the image source is reduced compared to the sensor.
kP1CaptureImageProperty_SourceHeight	UInt32	Read	Height of the image source in pixels.
kP1CaptureImageProperty_SourceActiveWidth	UInt32	Read	Width of the active area on the image source in pixels. The active area is the area on the image source where it is exposed.
kP1CaptureImageProperty_SourceActiveHeight	UInt32	Read	Height of the active area on the image source.
kP1CaptureImageProperty_SourceActiveXOffset	UInt32	Read	X-offset of the active area on the image source.
kP1CaptureImageProperty_SourceActiveYOffset	UInt32	Read	Y-offset of the active area on the image source.
kP1CaptureImageProperty_SourceOrientation	Int32	Read	Orientation of the image source relative to the device in degrees (0, 90, 180, 270). The angle is measured clockwise (to the right) from vertical.
kP1CaptureImageProperty_WhiteBalanceRed	Float64	Read	White balance red value (0.0 to 1.0).
kP1CaptureImageProperty_WhiteBalanceGreen	Float64	Read	White balance green value (0.0 to 1.0).
kP1CaptureImageProperty_WhiteBalanceBlue	Float64	Read	White balance blue value (0.0 to 1.0). The white balance is only updated on the device after writing the blue value.
kP1CaptureImageProperty_CameraOrientationMode	Enum	Read	Obsolete. Use *kCaptureImageProperty_CameraOrientationMode* instead.
kP1CaptureImageProperty_SensorPlus	Enum	Read	The Sensor+ mode the image was captured in.
kP1CaptureImageProperty_IntegrationTime	UInt32	Read	The time in milliseconds that the sensor was active during image capture.
kP1CaptureImageProperty_WhiteBalanceMode	Enum	Read	The white balance mode the image was captured in (auto, flash, fluorescent, etc.)
kP1CaptureImageProperty_HardwareGain	UInt32	Read	Phase One specific hardware gain.
kP1CaptureImageProperty_SoftwareGain	UInt32	Read	Phase One specific software gain.
kP1CaptureImageProperty_PreCompressionGain	UInt32	Read	Phase One specific pre-compression gain.
kP1CaptureImageProperty_ProcessingFlags	UInt32	Read	Phase One specific flags.
kP1CaptureImageProperty_BlackIntegrationTime	UInt32	Read	The time in milliseconds that the sensor was active during black calibration.
kP1CaptureImageProperty_BlackTemperature	Float64	Read	The sensor temperature during the black calibration in degrees Celsius.

Property	Typical Value Type	Typical Access	Description
kP1CaptureImageProperty_BlackTimeStamp	UInt64	Read	Timestamp for the black calibration, in seconds since January 1st, 1970 0:00.