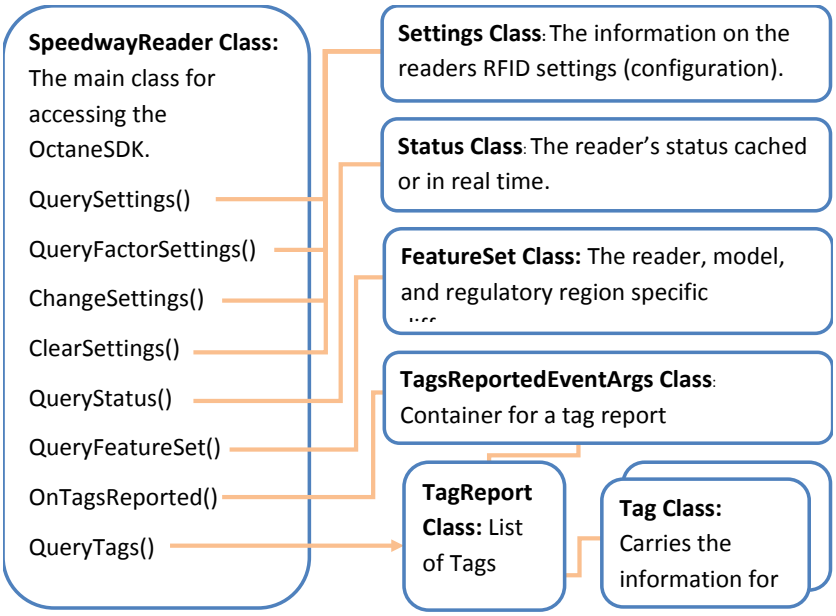
Octane SDK Quick Reference Guide

The OctaneSdk Quick Reference Guide is organized into tables covering Properties, Methods, and Callbacks for the main classes in the OctaneSdk. See the OctaneSdk Tutorial Workbook or Online Reference Guide for complete class descriptions and detailed programming information.



Speedway Reader Class

The Speedway Reader class is the main class in the OctaneSdk. Create instances of this class for each Reader you wish to control. Inherit from this class to add reader functionality to your program.

Properties	Description
ReaderIdentity	An Object in C#. A void* in C++. This value is set by the application and may be used as a back pointer to application specific data. The default is a hex string with the reader's unique identity (see *FeatureSet.ReaderIdentity*).
LogLevel	enum. The amount of detail the application requires logged from this *SpeedwayReader*. Example: *LogLevel.Warning*. Possible values are *Off, Error, Warning, Information, Verbose, Trace, and Debug*.

Methods	Description
void Connect string (readerName)	Connect to a Reader. The *readerName* is a string containing the host name or IP address of the Reader. After connection, either the *ChangeSettings* or *ResumeEventsAndReports* must be called before events or tag reports are delivered.
void Disconnect ()	Disconnect from the Reader.
void ResumeEventsAndReports ()	Called to resume control after reconnecting to a reader. Until this is called the events (notifications) and tag reports are held on the reader. This gives the application a chance to prepare. Note when this is called, events may be delivered that occurred in the past.
FeatureSet QueryFeatureSet ()	Return the *FeatureSet* of the Reader includes supported modes, powers, frequencies, and optional features.
Settings QuerySettings()	Return current Reader settings.
Settings QueryFactorySettings ()	Return Reader factory Settings. This is derived from the factory settings on the Reader itself and includes powers, frequencies, and modes. Factory settings vary by model and region.
void ClearSettings ()	Clear the Reader Settings. The Reader will not perform RFID operations.
void ChangeSettings (Settings settings)	Change the Reader Settings to the previous settings. Current Reader settings are first cleared, and then wholly replaced according to the settings specified. Once done the Reader can complete RFID operations.

Octane SDK Quick Reference Guide

Methods	Description
Status QueryStatus (StatusRefresh refreshWhat)	Return the Reader status including antennas, RFID operations, and GPI/O. The *refreshWhat* argument indicates what should be refreshed from the Reader itself. Normally the status is maintained based on notifications from the Reader. Example: *refreshWha*t is *StatusRefresh.Everything.* Possible values are None, Everything, JustAntennas, and JustGpis.
Status QueryStatus ()	Equivalent to *QueryStatus(StatusRefresh.None)*.
void Start() *void Stop ()*	Start/Stop the Reader. This may be called even if the Reader is set to auto-start/stop.
TagReport QueryTags (seconds)	Start the Reader, pause for the number of seconds, and then stop the Reader. Return a tag report of tags singulated during the specified time. Seconds may be an integer or a floating point number.
TagReport QueryTags ()	Returns the tags currently in the Readers accumulator. This call may be used while the Reader is singulating or idle. This call only makes sense if the Reader is set to use *ReportMode.WaitForQuery* or *ReportMode.BatchAfterStop.*
void SetGpo (int portNumber, bool level)	Change a general purpose output (GPO). The *portNumber* is in the range 1-8. The level is true for high and false for low. This call may be used even if the Reader has no settings.
ProgramAccessPasswordResult ProgramAccessPassword (ProgramAccessPasswordParams)	Program the tag access password.
ProgramKillPasswordResult ProgramKillPassword (ProgramKillPasswordParams)	Program the tag kill password.
ProgramEpcResult ProgramEpc (ProgramEpcParams)	Program the tag EPC, optionally setting the access password, and locking the tag.
ProgramUserBlockResult ProgramUserBlock(ProgramUserBlockParams)	Program a portion of a tag's user memory.
ProgramUserMemoryResult ProgramUserMemory (ProgramUserMemoryParams)	Program all of a tag's user memory.
void Log(LogLevel level, string format, ...)	Include an application generated log message into the log stream.

Callbacks (return void)	Description
OnLogging (LoggingEventArgs)	Deliver a log message. The base class discards the message. All callback functions return void.
OnConnectionChange (ConnectionChangedEventArgs)	The status of the network connection to the Reader changed. The base class demultiplexes to *OnConnected, OnDisconnected*, or *OnConnectionLost.*
OnConnected (ConnectionChangedEventArgs)	The Reader network connection was established.
OnDisconnected (ConnectionChangedEventArgs)	The network connection was disconnected upon request.
OnConnectionLost (ConnectionChangedEventArgs)	The network connection unexpectedly dropped. The situation must be cleaned up by calling Disconnect.
OnConnectionAttempted (ConnectionAttemptedEventArgs)	Another application tried to connect to the Reader.
OnAntennaChanged (AntennaChangedEventArgs)	An antenna was connected or disconnected.
OnGpiChanged (GpiChangedEventArgs)	An enabled general purpose input (GPI) changed. The base class demultiplexes to *OnGpi1Changed*, *OnGpi2Changed, OnGpi3Changed*, or *OnGpi4Changed*.
OnGpiXChanged (GpiChangedEventArgs)	GPI X changed. X replaces GPI 1-4 (Example: *OnGpi1Changed*… *OnGpi4Changed*).
OnReportBufferOverflowed (ReportBufferOverflowedEventArgs)	The report buffer on the Reader could not hold more tag reports. The number of Reader held tag reports depends on the report options enabled. Use *QueryTags* to resolve.
OnReportBufferWarned (ReportBufferWarnedEventArgs)	The report buffer on the Reader is nearly full. The application should do a *QueryTags t*o resolve the situation soon.
OnStarted (StartedEventArgs)	The Reader started.
OnStopped (StoppedEventArgs)	The Reader stopped.
OnTagsReported (TagsReportedEventArgs)	This tag report delivery occurs for *ReportModes BatchAfterStop, Individual*, and *IndividualUnbuffered.*

TagReport Class

The *TagReport* class contains tag information obtained by the Reader. Access *TagReport* instances through the *SpeedwayReader* class methods, callbacks, and events.

Properties	Description
ReaderIdentity	An Object in C#. A void* in C++. This is the application set value on *SpeedwayReader.ReaderIdentity*. It may be used as a back pointer to application specific data. The default is a hex string with the reader's unique identity (see *FeatureSet.ReaderIdentity)*.
Timestamp	A Time object in C++. The time the report was generated. A DateTime in C#.
Tags	A collection of tags. In C#, this is a List<Tag>. In C++, this is a Tag* pointer to an array of length nTags.

Tag Class

The Tag class contains the information obtained by the Reader for a specific EPC value. Access Tag instances through the *TagReport* class.

Properties	Description
ReaderIdentity	An Object in C#. A void* in C++. This is the value set by the application on *SpeedwayReader.ReaderIdentity*. It may be used as a back pointer to application specific data. The default is a hex string with the Reader's unique identity (see FeatureSet.ReaderIdentity).
AntennaPortNumber	**int.** The antenna port number if Is*AntennaPortNumberPresent* is true.
ChannelInMhz	**double**. The channel of the tag read for the EPC conveyed in this instance if *IsChannelInMhzPresent*
Epc	**string.** The EPC value backscattered by the tag.
FirstSeenTime	A *DateTime* in C#. A Time object in C++. The time of the first tag read for the EPC conveyed in this instance if Is*FirstSeenTimePresent* is true.
IsAntennaPortNumberPresent	**boolean.** Value denoting whether an antenna port number is present in this instance.
IsChannelInMhzPresent	**boolean.** Value denotes whether *ChannelInMhz* attribute is valid.
IsFirstSeenTimePresent	**boolean.** Value denotes whether *FirstSeenTime* attribute is valid.
IsLastSeenTimePresent	**boolean.** Value denotes whether *LastSeenTime* attribute is valid.
IsPeakRssiInDbmPresent	**boolean.** Value denoting whether a peak rssi is present in the instance.
IsPhaseAngleInRadiansPresent	**boolean.** Value denotes whether *PhaseAngleInRadians* attribute is valid.
IsSeenCountPresent	**boolean.** Value denoting whether the *SeenCount* attribute is valid.
IsSerializedTidPresent	**boolean.** Value denoting whether the *SerializedTid* attribute is valid.
LastSeenTime	A *DateTime* in C#. A *Time* object in C++. The time of the last tag read for the EPC conveyed in this instance if *IsLastSeenTimePresent* is true.
PeakRssiInDbm	**double.** The largest RSSI (least negative) value represented by the tag reads conveyed in this report if *IsPeakRssiInDbmPresent* is true.
PhaseAngleInRadians	**double.** The RF phase angle in radians (relative to the transmit carrier) the tag read for the EPC conveyed in this instance if *IsPhaseAngleInRadiansPresen*t
Rank	**int.** The order (index) of the tag within the report list.
SeenCount	**int.** The number of tag reads of this EPC conveyed in this tag report if *IsSeenCountPresent* is true.
SerializedTid	**string.** The serialized TID for the EPC conveyed in this instance if *IsSerializedTidPresent* is true.

Settings Class

The Settings class carries the information on the readers RFID settings (configuration). Access Settings class instances through the *SpeedwayReader* class methods.

Properties	Description
Label	**string.** A short name, similar to a file name, for these settings.
Description	**string.** A short description of these settings.

Properties	Description
ReaderMode	**enum.** The Reader mode controls modulation and data rate. Not all modes are available on all models and regions. Example: *ReaderMode.AutoSetDenseReader*. Possible values are: *AutoSetDenseReade*r, *AutSetSingleReader, MaxThroughput, Hybrid, DenseReaderM4, DenseReaderM8, and MaxMiller.*
SearchMode	**enum.** The search mode controls whether tags are singulated repeatedly or once. Example *SearchMode.DualTarget.* Possible values are: *ReaderSelected*, *DualTarget, SingleTarget, SingleTargetWithSuppression*.
Session	**int.** The session the reader should use. Sessions have different persistence characteristics after the tag no longer receives power from the reader. Possible values are 0-3.
TagPopulationEstimate	**int.** An estimate of how many tags will be in the Reader's field of view at one time.
Filters	Controls which tags the Reader singulates and reports.
Filters.Mode	**enum.** How the Reader combines the two filters. This essentially produces a truth value, and if true a tag participates in the Reader's current round of singulation. Example: *TagFilterMode.OnlyFilter1*. Possible values are *None, OnlyFilter1, Filter1AndFilter2, Filter1OrFilter2.*
Filters.TagFilter1, TagFilter2	The two filters
Filters.TagFilter1.MemoryBank	**enum.** The memory bank on which the filter is applied. Example: *MemoryBank.Tid.* Possible values are: *Reserved, Epc, Tid, User.*
Filters.TagFilter1.BitPointer	**int**. The bit offset in the specified memory bank at which the tag mask begins. Possible values are 0 to 512.
Filters.TagFilter1.BitCount	**int.** The number of bits contained within the tag mask.
Filters.TagFilter1.TagMask	**string.** A hex string representing the bit pattern to match. Example: "E208".
Filters.TagFilter1.FilterOp	**enum.** How the mask is used with respect to *Filters.Mode*. Example: *TagFilterOp.Match*. Possible values are: *None, Match, NotMatch*.
Antennas[port]	Table of antenna settings. The antenna is selected by its port number in the range of 1-4.
Antennas[port].PortNumber	**int.** The port number of the antenna. This is advisory to make it easier to identify the port number when iterating the table with *foreach.*
Antennas[port].IsEnabled	**bool.** True means the antenna is used during singulation.
Antennas[port].TxPowerInDbm	**double.** The amount of transmit power to use on the antenna. Only use power levels the Reader supports, typically in the range of 10.00 to 30.00 dBm in 0.25 dBm steps.
Antennas[port].MaxRxSensitivity	**bool**. Whether the Reader's maximum sensitivity is used. When true the Reader reports all tags that it can successfully communicate with. False indicates that the Reader should only report tags that have a return signal strength specified by *RxSenstivityInDbm* (see next field).
Antennas[port].RxSensitivityInDbm	**int.** The minimum signal strength that must be received by the Reader from the tag for the tag to be included in the report. Only use sensitivities supported by the Reader, typically in the range of -80 to -20 dBm in 1 dBm steps.
TxFrequenciesInMhz[i]	**double.** This is a set of transmit frequencies the Reader should use. It is an error to have any elements in this list if the Reader is operating in a hopping region. In non-hopping regions and an empty list means the Reader selects the frequencies to use. Use only frequencies supported by the Reader. Typical values vary by region. Example: *TxFrequenciesInMhz.Add(866.9)*
ReducedPowerTxFrequencies InMhz[i]	**double.** This is a set of transmit frequencies the Reader should use reduced power rather than the power specified by *TxPowerInDbm.* This is only available in the FCC region and is used to mitigate spectrum interference. Example: *ReducedPowerTxFreqeunciesInMhz.Add(910.250)*
LowDutyCycle	Low duty cycle is used in situations where the field of view is empty most of the time, and from time to time tags transit the field of view. While the field of view is empty, the Reader reduces its activity to mitigate interference and power consumption.
LowDutyCyle.IsEnabled	**bool.** When true, the Reader periodically pauses singulation on an antenna if it has not detected tags. When false the Reader constantly (subject to starting and stopping) attempts to singulate tags even in the absence of any tags.
LowDutyCycle. EmptyFieldTimeoutInMs	**int.** Dictates how long the Reader will attempt to singulate tags after it has determined that there are no tags in the field of view. After this time the antenna will enter low duty cycle where it stops continuous singulation and checks or "pings" from time to time to see if tags entered to field of view. Possible values are 10 to 60000
LowDutyCycle. FieldPingIntervalInMs	**int.** Dictates how long singulation is paused once the Reader deems a field of view empty. After this time the Reader resumes singulation.
AutoStart	The Reader can automatically start based on a trigger. Even when an auto-start is set, the Reader may be explicitly started using the *SpeedwayReader.Start()* method.

Octane SDK Quick Reference Guide

Properties	Description
AutoStart.Mode	**enum.** Indicates whether and how the Reader should automatically start. Exampe: *AutoStartMode.Immediate*. Possible values are: None, Immediate, GpiTrigger, and Periodic. Note: Immediate means that the Reader should always start. As soon as the Reader receives the Settings it starts. Even if stopped, the Reader will immediately restart.
AutoStart.GpiPortNumber	**int.** When using *GpiTrigger* mode, this is the port number of a general purpose input. When it changes to the *GpiLevel* value the Reader starts. Possible values are 1-4. **Note** that the GPI port must be enabled in *Settings.Gpis[]*
AutoStart.GpiLevel	**bool.** True for high, false for low. When using *GpiTrigger* mode, and when the GPI changes to this level the Reader starts.
AutoStart.FirstDelayInMs	**int.** When using Periodic mode, this specifies how long the Reader delays after receiving the Settings and before it first starts. After that, the Reader will, if stopped, start according to *PeriodInMs*. Zero (0) means there is no initial delay. Possible values are 0 to 1000000000.
AutoStart.PeiodInMs	**int.** When using Periodic mode, this is how often the Reader tries to start if it is stopped. The period is carefully maintained, and is relative to the previous periodic start. It has nothing to do with when the Reader stops. Zero (0) means there is no periodic start, only one start after the *FirstDelayInMs*. Possible values are 0 to 1000000000.
AutoStop	The Reader can automatically stop based on a trigger. Even when an auto-stop is set, the Reader may be explicitly stopped using the SpeedwayReader.Stop() method.
AutoStop.Mode	**enum.** Indicates whether and how the Reader should automatically stop. Example: *AutoStopMode.Duration.* Possible values are: *None, GpiTrigger, Duration*.
AutoStop.GpiPortNumber	**int.** When using *GpiTrigge*r mode, this is the port number of a general purpose input. When it changes to the *GpiLeve*l value the Reader stops. Possible values are 1-4. **Note** that the GPI port must be enabled in *Settings.Gpis[]*
AutoStop.GpiLevel	**bool.** True for high, false for low. When using *GpiTrigger* mode, and when the GPI changes to this level, then the Reader stops.
AutoStop.DurationInMs	**int.** When using *GpiTrigger* mode, this is the longest the Reader will singulate even if the GPI does not change. A value of 0 means there is no time limit. When using Duration mode, this is period the Reader singulates.
Report	The report settings controls how often the Reader sends a report of tags singulated, and which optional fields are reported.
Report.Mode	**enum.** Controls how often reports are sent. Example: *ReportMode.Individual*. Possible values are: *WaitForQuery, Individual, IndividualUnbuffered, BatchAfterStop*.
Report.IncludePeakRssi	**bool.** Indicates that the RSSI (Return Signal Strength Indication) should be reported. On Readers that support it, Impinj higher resolution RSSI is reported. Otherwise standard, 1dBm resolution RSSI is reported.
Report.IncludeAntennaPortNumber	**bool.** Indicates that the port number of the antenna used to singulate the tag should be reported.
Report.IncludeFirstSeenTime	**bool.** Indicates that the time the tag was first singulated (seen) should be reported. The time is since report accumulation started.
Report.IncludeLastSeenTime	**bool.** Indicates that the time the tag was last singulated (seen) and should be reported. The time is after the *FirstSeenTime*. It best to request this field only if accumulated reports are being used: modes *WaitForQuery* or *BatchAfterStop.*
Report.IncludeSeenCount	**bool.** Indicates that the number of times the tag was singulated (seen) should be reported. It is best to request this field only if accumulated reports are being used: *WatiForQuery* or *BatchAfterStop*. For *Individual* and *IndividualUnbuffered* the “seen” count is always 1.
Report.IncludeChannel	**bool.** Indicates that the channel, in Mhz, should be reported.
Report.IncludePhaseAngle	**bool.** Indicates that the phase angle of the tag backscatter relative to the carrier should be reported. This information is only significant if the channel is known. Few apps can make use of this report field.
Report.IncludeSerializedTid	**bool.** Indicates that Serialized TID of Impinj Monza4 tags should be reported.
Gpis[port]	Table of GPI settings. The GPI is selected by its port number in the range 1-4.
Gpis[port].PortNumber	**int.** The port number of the GPI. This is advisory to make it easier to identify the port number when enumerating the table with foreach.
Gpis[port].IsEnabled	**bool.** Indicates that the Reader should monitor the GPI port and generate a notification (event) when the input changes. **Note**: only enable GPI ports that are actually connected to stable inputs. A floating GPI can generate notification storms.

Properties	Description
Gpis[port].DebounceInMs	**int.** On Reader models that support debounce, when a GPI changes this is the amount of time that the Reader ignores subsequent changes. If at the end of that time the GPI level is different another notification is generated.
KeepAlive	The Reader can send a keep-alive message at regular intervals. The OctaneSdk promptly sends an acknowledgement. This assures the Reader and application that the network connection is healthy when the Reader has no tags or events to send.
KeepAlive.IsEnabled	**bool.** Indicates the Reader should send keep-alive messages at regular intervals.
KeepAlive.PeriodInMs	**int.** The frequency at which the Reader sends keep-alive messages. Possible values 0 to 60000.
KeepAlive.LinkDownThreshold	**int.** On Reader models that support it (see *FeatureSet.IsLinkMonitoringAvailable)*, the number of missed keep-alive messages at which point the connection is declared down or lost.

Status Class

The Status class allows access to the Reader's status cached or in real time. Access Status class instances through the SpeedwayReader class methods.

Properties	Description
ReaderIdentity	An Object in C#. A void* in C++. This is the value set on *SpeedwayReader.ReaderIdentity* by the application. It may be used as a back pointer to application specific data. The default is a hex string with the Reader's unique identity (see *FeatureSet.ReaderIdentity*).
Connection	**enum.** The state of the network connection to the Reader. Example: *ApplicationConnectionState.Connected*. Possible values are *NotConnected, Connected, Disconnected, and ConnectionLost.*
OperationalState	**enum.** The state of the Reader RFID activity. Example: *OperationalState.Ready*. Possible values are: *Unknown, NotConfigured, Ready, and Singulating*.
TemperatureInCelsius	**int.** The on-board temperature of the Reader's power amplifier.
Antennas[port]	Table of antenna status. The antenna is selected by its port number in the range 1-4.
Antennas[port].PortNumber	**int.** The port number of the antenna. This advisory eases identifying the port number when enumerating the table with *foreach*.
Antennas[port].State	**enum.** The connection state of the antenna port. Example: AntennaConnectionState.Connected. Possible values are Unknown, NotApplicable, Connected, and NotConnected.
Antennas[port].IsEnabled	**bool.** True indicates the antenna is enabled in the current settings and in use during singulation.
Antennas[port].IsConnected	**bool.** Short-hand property that indicates the antenna state is connected. AntennaConnectionState.Connected.
Gpis[port]	Table of GPI status. The GPI is selected by its port number in the range 1-4.
Gpis[port].PortNumber	**int.** The port number of the GPI. This is advisory to ease identifying the port number when iterating the table.
Gpis[port].State	**enum.** The state of the GPI port. Example: *GpioState.High.* Possible values are: *Unknown, NotApplicable, Low, and High.*
Gpis[port].IsEnabled	**bool.** True indicates the GPI is enabled in the current settings. The GPI port is monitored and notifications (event) will be sent when the input level changes.
Gpos[port]	Table of GPO settings. The GPO is selected by its port number in the range 1-8.
Gpos[port].PortNumber	**int.** The port number of the GPO. This is advisory to ease identifying the port number when enumerating the table with *foreach*.
Gpos[port].State	**enum.** The state of the GPO port. Example: *GpioState.High.* Possible values are: *Unknown, NotApplicable, Low,* and *High*.
IsConnected	**bool.** Short-hand property that indicates the network connection to the Reader is present and healthy.
IsSingulating	**bool.** Short-hand property that indicates the Reader is actively singulating tags.

FeatureSet Class

The FeatureSet class allows access to the Reader, model, and regulatory region specific differences. Access FeatureSet instances through the SpeedwayReader class methods

Properties	Description
ModelName	**string.** Example: Speedway R420
SerialNumber	**string**. Example: 403-06-20-0014
SoftwareVersion	**string**. The version of the Octane software. Example: 4.4.0.240
FirmwareVersion	**string.** The version of the modem firmware. Example: 4.4.0.240
FpgaVersion	**string.** The version of the FPGA program: Example: 4.4.0.240
PcbaVersion	**string**. The version of the Printed Circuit Board Assembly. Example: 250-002-000
ModelNumber	int. The numeric model number. Example: 2001002
Subregion	**string.** The regulatory rules the Reader is operating under. Example: US_FCC_Part_15
IsHoppingRegion	**bool**. True for hopping regions like FCC, false for regions where the TxFrequency may be specified like ETSI.
ReaderIdentity	**string**. A hex string representing the 64-bit identity derived from the Reader's Ethernet address. Example: 001625FFFF0000C3
AntennaCount	**int.** The number of antenna ports on the Reader. Example: 4
GpiCount	**int.** The number of General Purpose Input (Gpi) pins on the Reader. Example: 4
GpoCount	**int.** The number of General Purpose Output (Gpo) pins on the Reader. Example: 4
MaxAccessSpecs	**int.** The maximum number of accesses the Reader can track at one time. Example: 64
IsTagAccessAvailable	**bool.** True indicates the Reader can access (read/write/lock/kill).
IsFilteringAvailable	**bool.** True indicates tag inventory filtering is available.
IsGpiDebounceAvailable	**bool.** True indicates that GPI debounce is settable. Speedway R1000 Readers do not do debounce.
IsLinkMonitorAvailable	**bool.** True indicates that the Reader can automatically drop the connection to the application if it appears there is a network or application problem. This allows other hosts to connect to the Reader.
IsMultiwordBlockWriteAvailable	**bool.** True indicates that multiword block write to tags is available.
IsPhaseAngleReportingAvailable	**bool.** True indicates that the phase angle of a tag's backscatter can be reported.
IsSerializedTidReportingAvailable	**bool.** True that the Impinj serialized TID can be reported.
IsImpinjRssiReportingAvailable	**bool.** True indicates that higher resolution RSSI is available.
ReaderModes	Information about the Reader modes available. A Reader mode controls the bit rates, modulation, and other air protocol particulars. Not all modes are available on all models or in all regions.
ReaderModes.Entries[i]	The table of Reader modes available. On C++, *.nEntries* is the number of entries in the table. On C#, use *Entries.Count.*
ReaderModes.Entries[i].Description	**string.** The preferred description for the Reader mode.
ReaderModes.Entries[i].Mode	**enum**. The mode used in Settings by the application. Example: *ReaderMode.AutoSetDenseReader.* Possible values are: *AutoSetDenseReade*r, *AutSetSingleReader, MaxThroughput, Hybrid, DenseReaderM4, DenseReaderM8*, and *MaxMiller*.
ReaderModes.Entries[i].LlrpCode	**int.** An internal code. Generally not useful to apps programmers.
SearchModes	Information about the search modes available. Search modes control whether tags are singulated and reported repeatedly or once. Not all search modes are available on all versions of Reader software.
SearchModes.Entries[i]	The table of search modes available. On C++, .nEntries is the number of entries in the table. On C#, use Entries.Count.
SearchModes.Entries[i].Description	**string.** The preferred description for the search mode.
SearchModes.Entries[i].LlrpCode	**int.** An internal code. Generally not useful to apps programmers.
SearchModes.Entries[i].Mode	**enum.** The mode used in Settings by the application. Example *SearchMode.DualTarget.* Possible values are: *ReaderSelected, DualTarget, SingleTarget,* and *SingleTargetWithSuppression*.

Properties	Description
TxPowers	Information about the transmit powers available on the Reader. Typically transmit power is in the range of 10.00 to 30.00 dBm in 0.25 dBm steps. More power improves tag communication, but also increases may interference.
TxPowers.Entries[i]	The table of transmit powers available. On C++, *.nEntries* is the number of entries in the table. On C#, use *Entries.Count.*
TxPowers.Entries[i].Dbm	double. An available transmit power in dBm. Example: 28.25.
TxPowers.Entries[i].LlrpCode	**int.** An internal code. Generally not useful to apps programmers.
RxSensitivities	Information about receive sensitivities available on the Reader. Signals returned by tags below this level are ignored. Typical sensitivities are in the range of -80 to -20 dBm in 1 dBm steps
RxSensitivities.Entries[i]	The table of receive sensitivities available. On C++, *.nEntries* is the number of entries in the table. On C#, *Entries.Count.*
RxSensitivities.Entries[i].Dbm	**int.** An available receive-sensitivity in dBm. Example: -54
RxSensitivities.Entries[i].LlrpCode	**int.** An internal code. Generally not useful to apps programmers.
TxFrequencies	Information about the transmit frequencies used by the Reader. If the *IsHoppingRegion* is true these frequencies are advisory to the application. If *IsHoppingRegion* is false these are the frequencies that may be selected by the application. Note that in a hopping region the order of entries in this table is random.
TxFrequencies.Entries[i]	The table of transmit frequencies. On C++, *.nEntries* is the number of entries in the table. On C#, use *Entries.Count.*
TxFrequencies.Entries[i].Mhz	double. A transmit frequency in Mhz. Example: 913.25.
TxFrequencies.Entries[i].LlrpCode	**int.** An internal code. Generally not useful to apps programmers.
ReducedPowerTxFrequencies	Information about available frequencies that may have reduced power. On some Readers, the power of some frequencies may be reduced to mitigate interference.
ReducedPowerTxFrequencies .Entries[i]	The table of transmit frequencies. On C++, *.nEntries* is the number of entries in the table. On C#, use *Entries.Count.* An empty table (count is zero) means that no frequency powers may be reduced.
ReducedPowerTxFrequencies .Entries[i].Mhz	**double**. A transmit frequency in Mhz. Example: 913.25.
ReducedPowerTxFrequencies .Entries[i].LlrpCode	**int.** An internal code. Generally not useful to apps programmers.

Additional Support

The Impinj Learning Center provides application notes and code samples.

http://learn.impinj.com

Impinj Support provides technical support.

http://support.impinj.com