

ROConnect COM Reference

version 1.1

Table of Contents

Conventions	4
Interfaces.....	4
ITLP	4
IParameterValue	5
ICallback.....	6
IWaitableCallback extends ICallback.....	6
ITimeSyncCallback extends ICallback.....	7
IConnectionManager.....	7
IConnection.....	8
ISystemInformation	9
IParameterReader.....	10
IParameterWriter.....	11
IEventReader.....	12
IAlarmReader.....	13
IHistoryReader.....	13
IHistorySegment	14
IHistoryPoint.....	14
IHistoryIndexSet.....	16
IUserPointDefintions	16
IUserPointDefinition.....	16
IUserParameterDefinition.....	17
IEvent.....	17
ICalibrateVerifyEvent.....	17
IClockSetEvent	18
IFstEvent.....	18
IParameterChangeEvent	18
IPowerLostEvent.....	19
ISystemEvent	19
IUserEvent	19
IAlarm.....	20
IFstAlarm	20
IParameterAlarm.....	20
IUserTextAlarm	21
IUserValueAlarm.....	21
Enumerations	21
AccumulateRate.....	21
AlarmCode.....	21
AlarmType	21
ArchiveType	21
AveragingType	22
CommStatus.....	22
EventCode.....	22
EventType.....	22
HistoryType	22

LogicalCompatibilityMode.....	22
PointTypeLimits	22
RocDataType	22
RocDeviceType.....	22
TimeSyncResult.....	22

Conventions

The table below describes the data types used throughout the reference.

Type	Description
boolean	A true/false value
byte	An unsigned 8-bit integer
char	A signed 8-bit integer
word	An unsigned 16-bit integer
short	A signed 16-bit integer
dword	An unsigned 32-bit integer
integer	A signed 32-bit integer
time	An unsigned 32-bit integer representing a timestamp as the number of seconds since midnight, January 1 st , 1970, local time.
float	A 32-bit IEEE floating point number
double	A 64-bit IEEE floating point number
string	A unicode string
<type> ref	A reference or pointer to a variable of type <type>
<type> array	An array whose elements are of type <type>
<name> enum	An enumeration named <name>

Table 1 - Data Types

Property and parameters will be listed in the format “name:type [(in|out|in, out)]”, where name is the property or variable name, type is its data type, and the bracketed text indicates whether the item is read from or written to. out is for readable items, and in is for writable items.

Example: Count:dword [out]

Methods will be listed in the format “name(param1, param2, ..., paramN):type”, where name is the method name, paramX is a parameter description, and type is the return data type. If no return type is listed, the method has no return value.

Example: OpenFile(filename:string [in]):boolean

Interfaces

All of the interfaces described in this section extend IUnknown.

ITLP

This interface defines a class used to access TLP address information.

Properties

Type:byte [in, out]

The point type component of the address.

Logical:byte [in, out]

The logical point number component of the address

Parameter:byte [in, out]
The parameter number component of the address

Methods

Set(t:byte [in], l:byte [in], p:byte [in])
Returns a string representation of the value contained by the class.
t: The point type component of the address.
l: The logical point number component of the address
p: The parameter number component of the address

Coclasses

TLP

IParameterValue

This interface defines a class which encapsulates a value compatible with a ROC point parameter. Any value which can be stored in a ROC point parameter can be stored in a class implementing this interface.

Properties

Type:RocDataType enum [out]
The data type of the value currently contained in the class.

Size:dword [out]
The size of the value, in bytes.

Data:BYTE array [out]
The raw data that constitutes the value.

UINT8:byte [in, out]
The value. The return is undefined if Type \neq UINT8_TYPE. Type is automatically set to UINT8_TYPE when written to.

UINT16:word [in, out]
The value. The return is undefined if Type \neq UINT16_TYPE. Type is automatically set to UINT16_TYPE when written to.

UINT32:dword [in, out]
The value. The return is undefined if Type \neq UINT32_TYPE. Type is automatically set to UINT32_TYPE when written to.

INT8:char [in, out]
The value. The return is undefined if Type \neq INT8_TYPE. Type is automatically set to INT8_TYPE when written to.

INT16:short [in, out]
The value. The return is undefined if Type \neq INT16_TYPE. Type is automatically set to INT16_TYPE when written to.

INT32:integer [in, out]
The value. The return is undefined if Type \neq INT32_TYPE. Type is automatically set to INT32_TYPE when written to.

Float:float [in, out]

The value. The return is undefined if `Type ≠ FLOAT_TYPE`. `Type` is automatically set to `FLOAT_TYPE` when written to.

`Double:double [in, out]`

The value. The return is undefined if `Type ≠ DOUBLE_TYPE`. `Type` is automatically set to `DOUBLE_TYPE` when written to.

`String:string [in, out]`

The value. The return is undefined if `Type ≠ STRING_TYPE`. `Type` is automatically set to `STRING_TYPE` when written to.

`TLP:ITLP ref [in, out]`

The value. The return is undefined if `Type ≠ TLP_TYPE`. `Type` is automatically set to `TLP_TYPE` when written to.

Methods

`ConvertToString():string`

Returns a string representation of the value contained by the class.

Coclasses

`ParameterValue`

ICallback

This interface defines a class used to send notification of the completion of an asynchronous operation. Most long-running operations allow the caller to pass in a reference to an object that implements `ICallback`. This may be one of the provided coclasses (`TrivialCallback`, `WaitableCallback`, etc.), or the developer may choose to implement the interface with a class that meets their specific needs.

Methods

`Execute(caller:IUnknown ref [in], result:dword [in])`

Called by the object processing the asynchronous action when the operation is complete.

`caller`: A reference to the object which was processing the operation.

`result`: A result code from the calling object.

Coclasses

`TrivialCallback`, `WaitableCallback`

IWaitableCallback extends ICallback

This interface defines a class which will block until a specific number of operations have completed.

Properties

`Operations:dword [in, out]`

The number of operations to wait on.

Methods

`Wait(timeout:dword)`

This method will block until all the specified number of operations have completed, or until it has waited the number of milliseconds specified in the timeout, whichever comes first.

`timeout`: The maximum length of time to block, in milliseconds.

Coclasses

`WaitableCallback`

ITimeSyncCallback extends ICallback

This interface defines a class used to report back the results of a time synchronization operation. There is no coclass provided for this interface. The developer must provide their own implementation.

Properties

`Latency:word` [in, out]

The measured average network latency to the ROC.

`SD:word` [in, out]

The standard deviation of the network latency values.

`SyncTime:time` [in, out]

The time at which time synchronization occurred

`TimeDiff:dword` [in, out]

The difference between ROC and host times after synchronization, in milliseconds.

IConnectionManager

This interface defines a class which creates connections to a specific ROC, and limits the number of open connections to that ROC.

Properties

`BindAdapter:string` [in, out]

The IP address of the network adapter to use on the host machine.

`HostGroupId:byte` [in, out]

The group number for the host machine.

`HostUnitId:byte` [in, out]

The unit number of the host machine.

`IpAddress:string` [in, out]

The IP address of the ROC.

`PortNumber:word` [in, out]

The port number that the ROC is listening on.

`GroupId:byte` [in, out]

The group number of the ROC.

`UnitId:byte` [in, out]

The unit number of the ROC.

`UserId:string` [in, out]

The user name to use when logging into the ROC.

`Password:word` [in, out]

The password to use when logging into the ROC.

`MaxConnections:dword` [in, out]

The maximum number of open connections allowed to the ROC.

Retries:dword [in, out]

The number of retries to attempt if an error occurs when sending a command to the ROC.

RetryInterval:dword [in, out]

The base interval to wait between retry attempts.

CommandCount:dword [out]

The total number of commands sent to the ROC.

FailureCount:dword [out]

The total number of failures that have occurred communicating with the ROC.

LatencyAverage:double [out]

A running average of the network latency to the ROC.

LatencySD:double [out]

The standard deviation of the network latency samples.

Methods

CreateConnection():IConnection ref

Creates and initializes a connection to the ROC being managed by the manager.

Coclasses

ConnectionManager

IConnection

This interface defines an object used to communicate with a specific ROC.

Properties

AutoDisconnect:boolean [in, out]

If true, the connection will be closed after each operation.

LastError:string [out]

A description of the last error which occurred.

Manager:IConnectionManager ref [out]

The connection manager that manages the connection object.

Methods

Connect():boolean

Opens the connection to the ROC. Returns false if the connection cannot be made.

Disconnect()

Closes the connection to the ROC.

CreateAlarmReader():IAlarmReader ref

Creates an object which can be used to read the ROC's alarms.

CreateEventReader():IEventReader ref

Creates an object which can be used to read the ROC's event.

CreateHistoryReader():IHistoryReader ref

Creates an object which can be used to read the ROC's history points.

CreateParameterReader():IParameterReader ref

Creates an object which can be used to read the ROC's point parameters.

CreateParameterWriter():IParameterWriter ref

Creates an object which can be used to write to the ROC's point parameters.

GetSystemInformation():ISystemInformation ref

Creates an object which provides system information.

GetUserDefinedPoints():IUserDefinedPoints ref

Creates an object which can be used to read the ROC's user point definitions.

SynchronizeTime(
 cb:ICallback ref [in],
 reads:dword [in],
 maxLatency:dword [in],
 maxVariance:dword [in],
 tolerance:dword [in])

Initiates time synchronization.

cb: A reference to a callback object to be executed on completion. Passing a null reference causes the call to block.

reads: The number of reads to perform to measure network latency before synchronization.

maxLatency: The maximum allowed average latency. If the statistic exceeds this value, synchronization will not be performed.

maxVariance: The maximum allowed standard deviation of the network latency. If the statistic exceeds this value, synchronization will not be performed.

tolerance: The maximum allowed difference between the host and ROC times after synchronization. If the difference exceeds this value, an error value will be returned.

Coclasses

Connection

ISystemInformation

This interface defines a class that provides access to the ROC's system information.

Properties

DeviceType:RocDeviceType enum [out]

The model of ROC.

CompatibilityMode:LogicalCompatibilityMode enum [out]

Defines the number of slots, and points per slot.

OperatingMode:byte [out]

The mode the ROC is in. 0 = firmware update mode; 1 = run mode.

Revision:byte [out]

The version of the ROC Plus Protocol supported by the ROC.

RequestPort:word [out]

The port number on which the system information request opcode was received.

PointsPerSlot:byte [out]

The maximum number of points the ROC supports per slot.

FirstPointType:byte [out]

The number of the first supported point type.

LastPointType:byte[out]

The number of the last supported point type.

`PointCount:byte array [out]`

The number of points installed for a given point type. Array location N holding the number of points for point type N.

Methods

`Refresh()`

Refreshes the system information by retrieving it from the ROC.

Coclasses

`SystemInformation`

IParameterReader

This interface defines a class used to read point parameter values from the ROC.

Properties

`Connection:IConnection ref`

A reference to the connection object used to communicate with the ROC.

Methods

`AddParameter(`

`t:byte [in],`

`l:byte [in],`

`p:byte [in])`

Adds a single parameter to the read list.

`t`: The point type component of the parameter address.

`l`: The logical point component of the parameter address.

`p`: The parameter index component of the parameter address.

`AddParameters(params:ITLP ref array [in])`

Adds multiple parameters to the read list.

`params`: An array of references to TLP objects specifying the parameter addresses.

`AddPoint(`

`t:byte [in],`

`l:byte [in])`

Adds all of a point's parameters to the read list.

`t`: The point type component of the parameter address.

`l`: The logical point component of the parameter address.

`AddPoints(points:ITLP ref array [in])`

Adds all parameters from multiple points to the read list.

`points`: An array of reference to ITLP objects specifying the logical points. Only the T and L components of the TLPs are used.

`Initialize()`

Finalizes the list of requested parameters by generating opcodes.

`Read(cb:ICallback ref [in]):boolean`

Reads the requested parameters from the ROC. The read list is preserved, so subsequent calls will reread the same set of parameters unless `Reset` is called. Returns true if the operation succeeds. If `Initialize` has not been called, `Read` will do so for you.

`cb`: A reference to a callback object to notify when the operation completes. If a null reference is passed, the operation will block.

```
GetFirstValue(  
    tlp:ITLP ref ref [in],  
    value:IParameterValue ref ref [in])
```

Passes back the first value in the sequence read from the ROC.

`tlp`: A reference to a reference of an object to hold the address of the parameter. The object reference will be set to null if there are no values to return.

`value`: A reference to a reference of an object to hold the value of the parameter. The object reference will be set to null if there are no values to return.

```
GetNextValue(  
    tlp:ITLP ref ref [in],  
    value:IParameterValue ref ref [in])
```

Passes back the next value in the sequence read from the ROC.

`tlp`: A reference to a reference of an object to hold the address of the parameter. The object reference will be set to null if there are no more values to return.

`value`: A reference to a reference of an object to hold the value of the parameter. The object reference will be set to null if there are no more values to return.

```
GetValue(  
    t:byte [in],  
    l:byte [in],  
    p:byte [in]):IParameterValue ref
```

Retrieves a the value of a specific parameter. A null reference is returned if no value has been read for the requested parameter.

`t`: The point type component of the parameter address.

`l`: The logical point component of the parameter address.

`p`: The parameter index component of the parameter address.

```
Reset()
```

Clears the request list.

Coclass

ParameterReader

IParameterWriter

This interface defines a class used to write point parameter values to the ROC.

Properties

```
Connection:IConnection ref
```

A reference to the connection object used to communicate with the ROC.

Methods

```
SetValue(  
    t:byte [in],
```

```
l:byte [in],
p:byte [in],
value:IParameterValue ref [in])
```

Adds a single value to the write list. If a value is set for the same parameter more than once, only the last value will be written.

t: The point type component of the parameter address.
l: The logical point component of the parameter address.
p: The parameter index component of the parameter address.
value: The value to write.

```
SetValues(
    tlps:ITLP ref array [in],
    values:IParameterValue ref array [in])
```

Adds multiple values to the write list. If a value is set for the same parameter more than once, only the last value will be written.

tlps: An array of references to ITLP objects specifying the write addresses.
values: An array of references to IParameterValue object specifying write values.

```
Write(cb:ICallback ref [in]):boolean
```

Writes the current set of values to the ROC. If new values are added while the write is in progress, they will be held for the next write. If a write is requested while a previous write is still in progress, that write will be performed immediately after the active write completes. Returns true if the operation succeeds.

cb: A reference to a callback object to notify when the operation completes. A null reference may be passed, but the operation will block.

Coclasses

ParameterWriter

IEventReader

This interface defines a class used to read events from the ROC.

Properties

```
Connection:IConnection ref
```

A reference to the connection object used to communicate with the ROC.

```
CurrentIndex:word
```

The index of the last event written on the ROC.

```
EventCount:dword
```

The number of events read from the ROC.

```
Event:IEvent ref array
```

An array of event objects representing the events read from the ROC.

Methods

```
Read(cb:ICallback ref [in]):boolean
```

Reads the events from the ROC. Returns true if the operation succeeds.

cb: A reference to a callback object to notify when the operation completes. If a null reference is passed, the operation will block.

Coclasses

EventReader

IAlarmReader

This interface defines a class used to read alarms from the ROC.

Properties

Connection: IConnection ref

A reference to the connection object used to communicate with the ROC.

CurrentIndex: word

The index of the last alarm written on the ROC.

AlarmCount: dword

The number of alarms read from the ROC.

Alarm: IAlarm ref array

An array of alarm objects representing the alarms read from the ROC.

Methods

Read(cb: ICallback ref [in]): boolean

Reads the alarms from the ROC. Returns true if the operation succeeds.

cb: A reference to a callback object to notify when the operation completes. If a null reference is passed, the operation will block.

Coclasses

AlarmReader

IHistoryReader

This interface defines a class used to read history data from the ROC.

Properties

Connection: IConnection ref

A reference to the connection object used to communicate with the ROC.

Segment: IHistorySegment ref array

An array of segment objects representing the history segment read from the ROC.

Methods

Read(cb: ICallback ref [in],
readConfiguration: boolean [in],
readPointData: boolean [in],
indices: IHistoryIndexSet ref [in]): boolean

Reads the history data from the ROC. Returns true if the operation succeeds.

cb: A reference to a callback object to notify when the operation completes. If a null reference is passed, the operation will block.

readConfiguration: Pass in true if the history segment and point configuration information should be read.

readData: Pass in true if the history values should be read.

`indices`: A reference to a set of starting indices to use when reading data. Only those history values for a point between the start index and the current index will be read. A null reference can be passed if data is not being read, or if all values should be read.

Coclasses

HistoryReader

IHistorySegment

This interface defines an object which represents a history segment.

Properties

`Number:byte` [out]

The segment index.

`Description:string` [out]

A description of the segment.

`LoggingEnabled:boolean` [out]

True if history is being collected for the segment.

`ContractHour:byte` [out]

The hour at which a new day begins.

`DailyEntryCount:word` [out]

The number of entries recorded per day, per point for the segment.

`DailyEntryIndex:word` [out]

The index of the last daily entry written on the ROC.

`PeriodicEntryCount:word` [out]

The number of periodic entries recorded per point for the segment.

`PeriodicEntryIndex:word` [out]

The index of the last periodic entry written on the ROC.

`PeriodicSampleRate:byte` [out]

The number of minutes between periodic entries.

`AvailableEntries:dword` [out]

The number of history entries available system-wide.

`MaxPointCount:word` [out]

The maximum number of points that can be configured for the segment.

`ConfiguredPointCount:word` [out]

The number of points configured in the segment.

`PointCount:word` [out]

The number of active history points for the segment.

`Point:IHistoryPoint` ref array

An array of point objects representing the history points.

Coclasses

IHistorySegment

IHistoryPoint

This interface defines a class which represents a history point.

Properties

Number:byte [out]

The point index.

TLP:ITLP ref [out]

The parameter being recorded in the point.

Description:string [out]

A description of the history point.

TagId:string [out]

The TAG parameter of the logical point that the target parameter is in.

Segment:IHistorySegment ref [out]

A reference to the segment that the point is a part of.

Type:ArchiveType enum [out]

The type of value being stored in the point.

Averaging:AveragingType enum [out]

The type of averaging used. Only valid for averaging archive types.

AccumRate:AccumulateRate enum [out]

The period of accumulation used. Only valid for accumulate archive types.

TodayMaxTime:time [out]

The timestamp at which the highest value of the day was recorded.

TodayMaxValue:float [out]

The highest value of the day.

TodayMinTime:time [out]

The timestamp at which the lowest value of the day was recorded.

TodayMinValue:float [out]

The lowest value of the day.

YesterdayMaxTime:time [out]

The timestamp at which the highest value of the day was recorded.

YesterdayMaxValue:float [out]

The highest value of the day.

YesterdayMinTime:time [out]

The timestamp at which the lowest value of the day was recorded.

YesterdayMinValue:float [out]

The lowest value of the day.

CurrentValue:float [out]

The current point value.

LastDailyValue:float [out]

The last value recorded as a daily entry.

Timestamp: time array array [out]

A 2-dimensional array of timestamps. The first dimension is indexed by history type (use the HistoryType enumeration), and the second dimension is the index of the desired entry within the history type.

PointValue:float array array [out]

A 2-dimensional array of point values. The first dimension is indexed by history type (use the HistoryType enumeration), and the second dimension is the index of the desired entry within the history type.

Coclasses

HistoryPoint

IHistoryIndexSet

This interface defines a set of starting indices for history reads. When passed to `IHistoryReader::Read`, only the history values after the index specified in the set will be read. The read will wrap around to the beginning of value list if the current index is lower than the start index specified.

Properties

`Index:word array array [in, out]`

A two dimensional array of starting indices. The first dimension of the array is the segment number, the second dimension of the array is the history type. Use the `HistoryType` enumeration to index in the second dimension.

Coclasses

`HistoryIndexSet`

IUserPointDefintions

This interface defines a class used to access the defintion of user defined points in the ROC.

Properties

`FirstPoint:IUserPointDefinition ref [out]`

The first definition in the sequence of user defined points. Returns a null reference if there are no user defined points in the ROC.

`NextPoint:IUserPointDefinition ref [out]`

The next definition in the sequence of user defined points. Returns a null reference if there are no more user defined points in the ROC.

`Point:IUserPointDefinition ref array [out]`

The user defined points, indexed by their point number. If there is no user defined point at the requested index, a null reference is returned.

Coclasses

`UserPointDefinitions`

IUserPointDefinition

This interface defines a class used to access the defintion of a user defined point in the ROC.

Properties

`TypeNumber:byte [out]`

The point type number.

`Name:string [out]`

The name of the point type.

`Abbrev:string [out]`

The abbreviation of the point type's name.

`LogicalCount:byte [out]`

The number of logical points.

ParameterCount:byte [out]

The number of parameters the point type has.

Parameter:IUserParameterDefinition ref array [out]

An array of references to parameter definitions.

Coclasses

UserPointDefinition

IUserParameterDefinition

This interface defines a class used to access the definition of a user defined point's parameters.

Properties

Id:byte [out]

The parameter index.

Name:string [out]

The name of the parameter.

Abbrev:string [out]

The abbreviation of the parameter's name.

Type:RocDataType enum [out]

The data type of the parameter.

Size:byte [out]

The size of the parameter data, in bytes.

ReadOnly:boolean [out]

True if the parameter is read only.

Coclass

UserParameterDefinition

IEvent

This interface defines a class used to represent a generic ROC event. In most cases an additional interface can be queried to obtain event-type specific information.

Properties

Timestamp:time [out]

The timestamp on the ROC when the event was logged

Type:EventType enum [out]

The type of event.

Coclasses

Event

ICalibrateVerifyEvent

This interface defines a class used to represent a calibrate verify ROC event.

Properties

OperatorId:string [out]

The operator that initiated the calibration.

TLP:ITLP ref [out]

The TLP of the parameter being calibrated.

Raw:float [out]

The raw value of the parameter after calibration.

Calibrated:float [out]

The calibrated value of the parameter after calibration.

Coclasses

CalibrateVerifyEvent

IClockSetEvent

This interface defines a class used to represent a clock set ROC event.

Properties

NewTime:time [out]

The time set to the clock.

Coclasses

ClockSetEvent

IFstEvent

This interface defines a class used to represent an FST ROC event.

Properties

FstNumber:byte [out]

The FST that triggered the event.

Description:string [out]

A text description of the event.

PointValue:float [out]

A values associated with the event.

Coclasses

FstEvent

IParameterChangeEvent

This interface defines a class used to represent a parameter change ROC event.

Properties

OperatorId:string [out]

The operator that changed the parameter value.

TLP:ITLP ref [out]

The TLP of the parameter whose value changed.

DataType:RocDataType enum [out]

The data type of the parameter whose value changed.

OldValue:IParameterValue ref [out]

The previous value of the parameter. This property will have a type of INVALID_TYPE if there was not enough room in the event record to store the old value.

newValue: IParameterValue ref [out]

The new value of the parameter. String values may be truncated if they were too large to fit in the event record.

Coclasses

ParameterChangeEvent

IPowerLostEvent

This interface defines a class used to represent a power lost ROC event.

Properties

LossTime: time [out]

The time at which power was lost.

Coclasses

PowerLostEvent

ISystemEvent

This interface defines a class used to represent a system ROC event.

Properties

Code: EventCode enum [out]

A code which specifies an event sub-type.

Description: string [out]

A text description of the event.

Coclasses

SystemEvent

IUserEvent

This interface defines a class used to represent a user ROC event.

Properties

OperatorId: string [out]

The operator that caused the event.

Code: EventCode enum [out]

A code which specifies an event sub-type.

Description: string [out]

A text description of the event.

Coclasses

UserEvent

IAlarm

This interface defines a class used to represent a generic ROC alarm. In most cases an additional interface can be queried to obtain alarm-type specific information.

Properties

Timestamp:time [out]

The timestamp on the ROC when the alarm was logged

Type:AlarmType enum [out]

The type of alarm.

IsSet:boolean [out]

True if the alarm is being set, false if it is being cleared.

SRBX:boolean [out]

True if this is a Spontaneous Report-By-Exception alarm.

Coclasses

Alarm

IFstAlarm

This interface defines a class used to represent an FST ROC alarm.

Properties

FstNumber:byte [out]

The FST that triggered the alarm.

Description:string [out]

A text description of the alarm.

PointValue:float [out]

A values associated with the alarm.

Coclasses

FstAlarm

IParameterAlarm

This interface defines a class used to represent a parameter ROC alarm.

Properties

Code:AlarmCode enum [out]

A code which specifies an alarm sub-type.

Description:string [out]

A text description of the alarm.

TLP:ITLP ref [out]

The TLP of the parameter that triggered the alarm.

ParameterValue:IParameterValue ref [out]

The value of the parameter when the alarm was triggered/cleared.

Coclasses

ParameterAlarm

IUserTextAlarm

This interface defines a class used to represent a user ROC alarm.

Properties

Description:string [out]
A text description of the alarm.

Coclasses

UserTextAlarm

IUserValueAlarm

This interface defines a class used to represent a user ROC alarm.

Properties

Description:string [out]
A text description of the alarm.
AlarmValue:IParameterValue ref [out]
A value associated with the alarm.

Coclasses

UserValueAlarm

Enumerations

AccumulateRate

DAY_RATE
HOUR_RATE
MINUTE_RATE
SECOND_RATE

OFF_SCAN_MODE_AC
MANUAL_FLOW_INPUTS_AC
METER_TEMP_FAIL_AC
COMPRESS_CALC_AC

AlarmCode

LOW_AC
LOW_LOW_AC
HIGH_AC
HIGH_HIGH_AC
RATE_AC
STATUS_CHANGE_AC
POINT_FAIL_AC
SCANNING_DISABLED_AC
SCANNING_MANUAL_AC
REDUNDANT_TOTAL_COUNTS_AC
REDUNDANT_FLOW_REGISTER_AC
NO_FLOW_AC
INPUT_FREEZE_MODE_AC
SENSOR_COMM_FAIL_AC
RS485_COMM_FAIL_AC

AlarmType

NO_ALARM
PARAMETER_ALARM
FST_ALARM
USER_TEXT_ALARM
USER_VALUE_ALARM

ArchiveType

NONE_ARCHIVE
USER_C_DATA_ARCHIVE
USER_C_TIME_ARCHIVE
FST_DATA_ARCHIVE
FST_TIME_ARCHIVE
AVERAGE_ARCHIVE
ACCUMULATE_ARCHIVE
CURRENT_VALUE_ARCHIVE
TOTALIZE_ARCHIVE

AveragingType

NONE_AVERAGE
FLOW_TIME_LINEAR_AVERAGE
FLOW_TIME_FORMULA_AVERAGE
FLOW_LINEAR_AVERAGE
FLOW_FORMULA_AVERAGE
LINEAR_AVERAGE

CommStatus

COMM_OK
COMM_BIND_FAILED
COMM_CONNECT_FAILED
COMM_POOL_EMPTY
COMM_LOGIN_FAILED

EventCode

INITIALIZATION_SEQUENCE
ALL_POWER_REMOVED
INITIALIZE_FROM_DEFAULTS
ROC_CRC_ERROR
DATABASE_INITIALIZATION
PROGRAM_FLASH
CLOCK_SET
TEXT_MESSAGE
DOWNLOAD_CONFIGURATION
CALIBRATION_TIMEOUT
CALIBRATION_CANCEL
CALIBRATION_SUCCESS
MVS_RESET_TO_DEFAULTS

EventType

NO_EVENT
PARAMETER_CHANGE_EVENT
SYSTEM_EVENT
FST_EVENT
USER_EVENT
POWER_LOST_EVENT
CLOCK_SET_EVENT
CALIBRATE_VERIFY_EVENT

HistoryType

MINUTE_HISTORY
PERIODIC_HISTORY
DAILY_HISTORY

LogicalCompatibilityMode

MAX_16_POINTS_9_SLOTS,
MAX_16_POINTS_15_SLOTS,
MAX_8_POINTS_27_SLOTS

PointTypeLimits

FIRST_POINT_TYPE
LAST_POINT_TYPE

RocDataType

BINARY_TYPE
INT8_TYPE
INT16_TYPE
INT32_TYPE
UINT8_TYPE
UINT16_TYPE
UINT32_TYPE
FLOAT_TYPE
TLP_TYPE
STRING3_TYPE
STRING7_TYPE
STRING10_TYPE
STRING12_TYPE
STRING20_TYPE
STRING30_TYPE
STRING40_TYPE
DOUBLE_TYPE
TIME_TYPE
INVALID_TYPE

RocDeviceType

ROCPAC_3XX_DEVICE
FLOBOSS_407_DEVICE
FLASHPAC_3XX_DEVICE
FLOBOSS_503_DEVICE
FLOBOSS_504_DEVICE
ROC_8XX_DEVICE

TimeSyncResult

SYNC_OK
SYNC_FAILED
EXCESSIVE_LATENCY
EXCESSIVE_VARIANCE
EXCESSIVE_DIFFERENCE