Security Management System

MANUAL

SMS API

VANDERBILT INDUSTRIES

Vanderbilt Industries Copyright Notice

CONTACT INFORMATION

Vanderbilt Industries
Phone: 855-316-3900
Fax: 973-316-3999
www.vanderbiltindustries.com

Contents

SMS Communication API

CHAPTER 1

Introduction

This document describes the SMS API which can be used to query the database and implement real time communication with the Vanderbilt Security Management System.

Installing the API

To install the API follow the instructions below:

1. Install SMS v6.0.0. See the SMS manual for installation details.
2. The SMS API files will be located in the SMS installation folder (typically c:\program files (x86)\SMS\)
3. Navigate to the SMS BIN folder.
4. Extract the contents of SMS_API_v1.50.zip into the development environment.
5. The 3 DLLs in the BIN folder comprise the SMS API and are all that is required.
6. The source code for a Visual Studio 2010 C# Windows Form API Demonstration Application is included in the SMS_API_Demonstration folder under the Help folder.
7. Compile the SMS_API_Demonstration solution to test SMS API functionality with the installed instance of SMS v6.0.
8. Obtain an Electronic License Key with the SMS API Enabled (*see the SMS Electronic License Key chapter for details*).

Structure

The SMS API contains several parts:

- A Win32 DLL that contains functions for basic real time messaging to and from SMS (SPComm.dll)
- A .NET assembly DLL that encapsulates the win32 functions into a .NET object, and also adds other features (SMS_API.dll and SPCommWrapper.dll together implementing the SMS namespace)
- A sample Visual Studio 2010 C# solution showing how to use the .NET assembly (SMS_API_demonstration.sln)
- This documentation file

SPComm.dll

The key features provided by the SPComm.dll:

- Ability to monitor transactions and other real time events
- Ability to insert transactions into the system
- Ability to send database change notices to the system
- Ability to determine the status of some devices in the system, notably contact points
- Ability to issue manual overrides and manual override sets
- Ability to acknowledge alarms
- Ability to detect the status of the connection to the System Processor

.NET SMS_API

The features added by the .NET assembly are

- Accessibility to the .NET environment
- Automatic database lookup of items like cardholder names, device descriptions, etc.
- Refactoring of setup and tear down code into a class
- Refactoring of callback functions into .NET delegates
- Refactoring of error codes into typed exceptions
- Automatic interpretation of many fields, such as the message type ID, into information more meaningful to client code

Where to start for .NET users...

See the SMS namespace and the SMS_API_Demonstration solution (which implements a simple real time log display and control application). Typically, clients instantiate a SMS:SMS_API object which will establish communications and register delegates to receive notifications such as AlarmHandler(), and then call the member functions such as ExecuteManualOverrideSet() as needed. Finally call SMS_API.Dispose() and set the object to nil on completion.

Caveats and errata

- The SPComm.dll file must be located same folder as the client executable, or in the system32 folder.
- Clients must turn off unicode and pure clr options within Visual Studio. These can be accessed through the project properties.
- Multibyte support should be enabled as well as normal common language runtime support (/clr rather than /clr:pure).
- References to the SMS_API and SPCommWrapperBase DLLs should be placed in any .NET projects that use the SMS namespace.
- The following entries must be appended to the Windows Services file (typically located in "c:\Windows\System32\Drivers\Etc\"):

geo_sp	5355 / tcp
geo_cm	5354 / tcp
geo_spxml	5359 / tcp
geo_rr	21380 / tcp

- If the message "Attempting managed execution inside OS Loader lock" is generated while disposing a reference to the DLL from managed code, disable the debugger warning/exception on this topic. Choose Debug->Exceptions from the menu (or ctrl-alt-e) and then uncheck the "Managed Debugging Assistants->LoaderLock->Thrown" checkbox.

SMS Electronic License Key

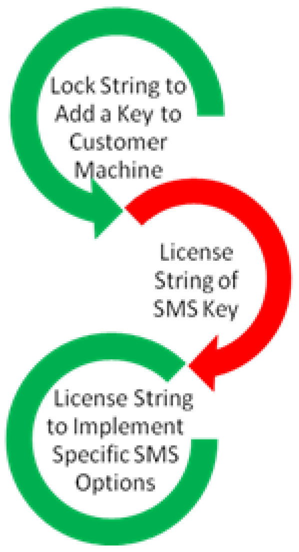
CHAPTER 2

Introduction

For the API to function, SMS V6.0.0 must be installed utilizing an Electronic License Key with the API enabled. This chapter details how to acquire and install an Electronic License Key if not already installed with SMS.

Overview

Sentinel SuperPro is a combination of a data structure, used as a key, and algorithms for securely creating, updating, and reading that data structure. The data structure (key) is bound to a piece of hardware which must interface with the computer in order for that computer to run SMS. That piece of hardware is a USB Key in previous versions of SMS. The process described herein will remove SMS reliance on the USB Key and create an Electronic License Key which is stored on the computer's hard drive, removing the need for the USB Key. The Electronic License is bound to a combination of the target computer's hardware and it ensures that the Electronic License Key will only work on one specific computer. The Electronic License Key is compatible with VMware virtual systems. It is **not** transferrable to another computer.


Lock String to Add a Key to Customer Machine
License String of SMS Key
License String to Implement Specific SMS Options

In order to create an Electronic License Key:

1. If SMS V6.0.0 has not yet been installed, install the SMS V6.0.0 files (see the SMS Software Manual for details) or the SMS Licensing Tools (available as a separate download).
2. Create a Lock Code for the server or workstation running the SMS System Processor (SP) and furnish the code to Vanderbilt Industries.
3. Vanderbilt Industries generates a unique License String from the Lock Code which will grant the customer's specific SMS options.
4. The Customer installs the License String onto the server or workstation running the SMS System Processor (SP).

Creating a Lock Code

To create a Lock Code, follow the instructions below:

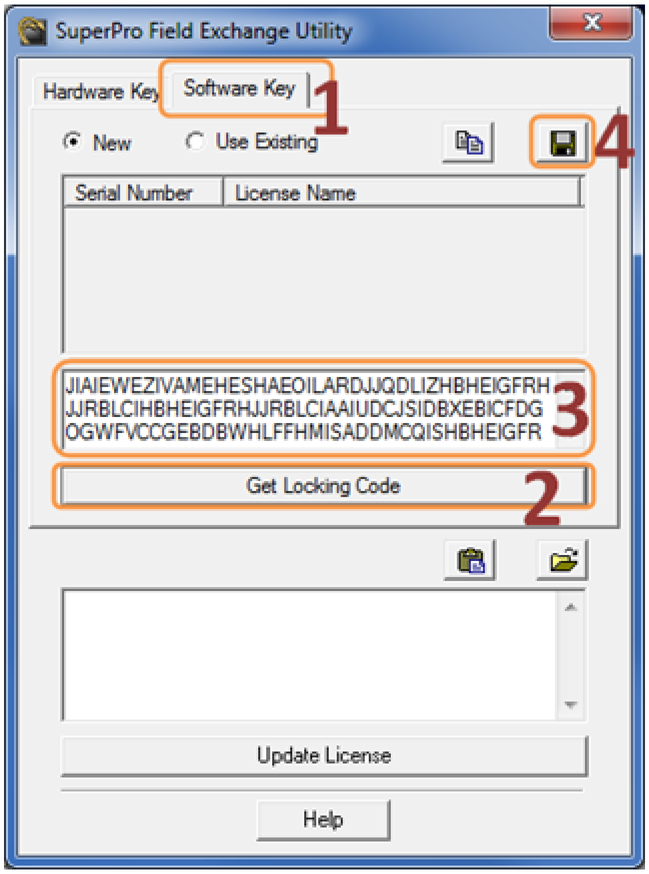
1 Open the Super Pro Field Exchange Utility.

a) Go to the Bin folder of SMS (*or the alternate location of FieldExUtil.exe*).

b) Go to the Sentinel folder.

c) Double Click on the FieldExUtil.exe icon. The SuperPro Field Exchange Utility will open.

Note: You may add the Field Exchange Utility to the SMS Launcher. See the SMS Software Manual for instructions on how to use System Security to add applications to the SMS Launcher.


SuperPro Field Exchange Utility
Hardware Key
Software Key
1
New
Use Existing
4
Serial Number
License Name
JIAIEWEZIVAMEHESHAEOILARDJJQDLIZHBHEIGFRH
JJRBLCIHBHEIGFRHJJRBLCIAAIUDCJSIDBXEBICFDG
OGWFVCCGEBDBWHLFFHMISADDMCQISHBHEIGFR
3
Get Locking Code
2
Update License
Help

2 Click the **Software Key** tab (**Label 1**).

3 Click the **Get Locking Code** button (**Label 2**). A **Locking Code** will be generated in the text field (**Label 3**).

4 Click the **Save icon** (**Label 4**) to save the Locking code to a .LOC file.

5 Send the .LOC file to SMS Electronic License Processing at SMSELicense@VanderbiltIndustries.com. A License File matching the .LOC file and enabling the specified SMS features will be prepared and sent back for installation. The license *will not be transferable to another computer*.

Installing the License File

Once you receive the License File (.LIC) from SMS Electronic License Processing it must be installed. Follow the instructions below to install the License File.

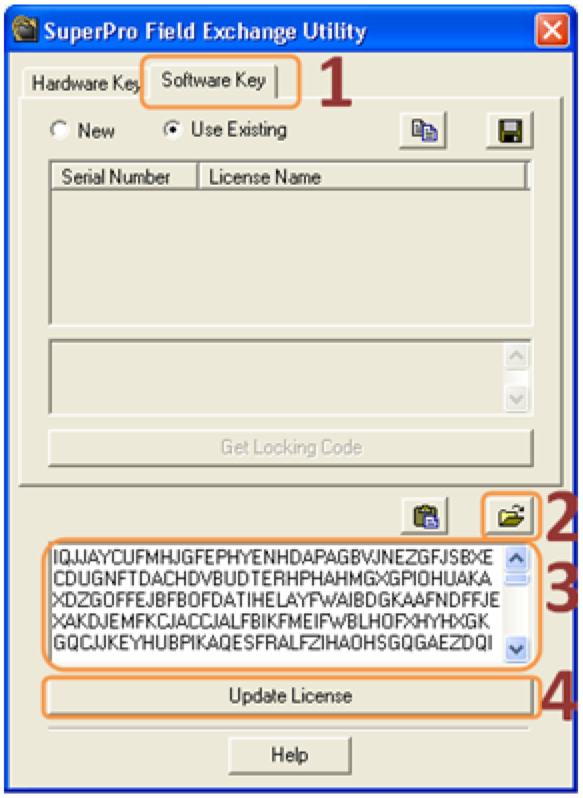
1 Open the Super Pro Field Exchange Utility.

a) Go to the **Bin** folder of SMS.

b) Go to the **Sentinel** folder.

c) Double Click on the **FieldExUtil.exe** icon. The SuperPro Field Exchange Utility will open.

Note: You may add the Field Exchange Utility to the SMS Launcher. See the SMS Software Manual for instructions on how to use System Security to add applications to the SMS Launcher.

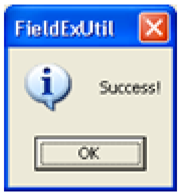


```
IQJJAYCUFMHJGFEPHYENHDAPAGBVJNEZGFJSBXE
CDUGNFTDACHDVBUDTERHPHAHMGXGPIOHUAKA
XDZGOFFEJBFBOFDATIHELAYFWAIBDGKAAFNDFFJE
XAKDJEMFKCJACCJALFBIKFMEIFWBLHOFXHYHXGK
GQCJJKEYHUBPIKAQESFRALFZIHAOHSGQGAEZDQI
```

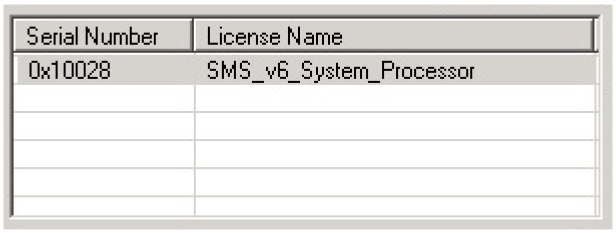
2 Click the **Software Key** tab (**Label 1**).

3 Click the **Load License Code** button (**Label 2**). The License Code will appear in the text field (**Label 3**).

4 Click the **Update License** button (**Label 4**). The FieldEXUtil window will open.



5 Click **OK**. The Serial Number and License Name will appear in the appropriate fields. The SMS System Processor Electronic License Key is now defined.



Serial Number	License Name
0x10028	SMS_v6_System_Processor

6 The SMS View SP Status application can also be used to verify that the Electronic License Key is recognized by SMS and will display the licensed features.

Class

CHAPTER 3

Class Hierarchy

- SMS::Abstract
- SMS.AlarmComment
- SMS.AlarmInstruction
- SMS.DBObject
    - SMS.AlarmCriteria
    - SMS.Area
    - SMS.AreaAccess
    - SMS.AreaSet
    - SMS.Cardholder
    - SMS.Credential
    - SMS.Device
    - SMS.DeviceType
    - SMS.ManualOverride
    - SMS.ManualOverrideSet
    - SMS.Operator
    - SMS.Transaction
    - SMS.Alarm
    - SMS.TransactionCode
    - SMS.TransactionGroup
- SMS.DeviceStatusMessage
    - SMS.ContactStatusMessage
    - SMS.ReaderStatusMessage
    - SMS.RelayStatusMessage
- SMS::FatalException
- SMS::NonFatalException
    - SMS::CIMCommunicationException
    - SMS::ControllerCommunicationException
    - SMS::DataNotLoadedException

    - SMS::DBCommunicationException
    - SMS::DLLAlreadyOpenException
    - SMS::DLLNotOpenException
    - SMS::InvalidParameterException
    - SMS::LibraryLoadingException
    - SMS::LicenseCountExceededException
    - SMS::LicenseCountNotRetrievedException
    - SMS::LicenseInvalidException
    - SMS::MemoryAllocationException
    - SMS::MemoryDeallocationException
    - SMS::SPCommunicationException
    - SMS::SPConnectionTimeoutException
    - SMS::SPLoginTimeoutException
    - SMS::UnsupportedOperationException
- SMS.ShuntStatusMessage
- SMS.SMS_API
- SMS.SMS_DB
- SMS.VideoCamera

Class List

Here are the classes, structures, unions and interfaces with brief descriptions:

- **SMS::abstract** - Clients should not instantiate this class. Use SMS::SMS_API
- **SMS.Alarm** - Transaction designated as an Alarm; requires Operator attention
- **SMS.AlarmComment** - Descriptive notes associated with an Alarm
- **SMS.AlarmCriteria** - Information about the criteria that caused the Alarm
- **SMS.AlarmInstruction** - Instructions are associated to the criteria generating an Alarm. Alarms generated by the same criteria share Instructions.
- **SMS.Area** - An Area is an abstract container that may have zero, one or many devices assigned
- **SMS.AreaAccess** - An AreaAccess record indicates access to a door
- **SMS.AreaSet** - An AreaSet is an abstract container that may have zero, one or many Areas assigned
- **SMS.Cardholder** - Cardholders are personnel interacting with card Reader Devices using security Badges
- **SMS::CIMCommunicationException** - Exception for signaling CIM communication errors
- **SMS.ContactStatusMessage** - Contact Device Status Message
- **SMS::ControllerCommunicationException** - Exception for signaling Controller communication errors
- **SMS.Credential** - Credentials are items (typically badges) presented to a Reader to gain access
- **SMS::DataNotLoadedException** - Exception for signaling that internal reference data required by SMS_API was not successfully loaded
- **SMS::DBCommunicationException** - Exception for signaling database communication errors
- **SMS.DBObject** - SMS_API objects with information stored in the SMS database inherit from this abstract class which provides the foundations for basic database I/O
- **SMS.Device** - Devices are things like Readers, Contacts, Workstations, etc.
- **SMS.DeviceStatusMessage** - Device Status Message
- **SMS.DeviceType** - Indicates the types of devices that can exist (Readers, Contacts, Workstations, etc.)
- **SMS::DLLAlreadyOpenException** - Exception for signaling SMS_API.dll already open errors
- **SMS::DLLNotOpenException** - Exception for signaling SchlagAPI.dll not open errors
- **SMS::FatalException** - Exception for signaling fatal errors
- **SMS::InvalidParameterException** - Exception for signaling that one or more parameters passed to a routine are invalid
- **SMS::LibraryLoadingException** - Exception for signaling library loading errors
- **SMS::LicenseCountExceededException** - Exception for signaling that SMS_API is licensed, but the total number of workstation and/or API logins exceeds the number of licensed client logins
- **SMS::LicenseCountNotRetrievedException** - Exception for signaling that an error occurred retrieving license count information from the SP
- **SMS::LicenseInvalidException** - Exception for signaling that SMS_API is not a licensed feature for this SMS installation

- **SMS.ManualOverride** - A ManualOverride (MRO) is used to initiate a Device action (e.g. open a door, energize a relay, suspend contact reporting, etc.)
- **SMS.ManualOverrideSet** - A ManualOverrideSet is a collection of MROs
- **SMS::MemoryAllocationException** - Exception for signaling that memory could not be allocated on the global heap)
- **SMS::MemoryDeallocationException** - Exception for signaling that memory could not be deallocated on the global heap
- **SMS::NonFatalException** - Exception for signaling non fatal errors
- **SMS.Operato**r - SMS application Operator
- **SMS.ReaderStatusMessage** - Reader Device Status Message
- **SMS.RelayStatusMessage** - Relay Device Status Message
- **SMS.SMS_API** - Primary Class for SMS Access Control System Interaction. Documentation for many important methods are found in parent class SPCommWrapperBase
- **SMS.SMS_DB** - This class abstracts database interaction and contains methods that return lists of DBObjects
- **SMS.ShuntStatusMessage** - Status Message explaining Report Shunting, Trigger Shunting or a Relay state
- **SMS::SPCommunicationException** - Exception for signaling SP communication errors
- **SMS::SPConnectionTimeoutException** - Exception for signaling that a timeout occurred waiting to connect to the SP
- **SMS::SPLoginTimeoutException** - Exception for signaling that a timeout occurred waiting to login to the SP
- **SMS.Transaction** - A Transaction is any event recorded by the system for future reporting and that may appear in the real time monitoring software
- **SMS.TransactionCode** - TransactionCode defines the type of transaction (i.e. a Valid Access Transaction versus a Contact Active transaction)
- **SMS.TransactionGroup** - Transaction Groups indicate a class of transactions, such as a Valid Access transactions versus Contact Active transactions
- **SMS::UnsupportedOperationException** - Exception for signaling that an operation is not supported (e.g. getting the status of a workstation Device
- **SMS.VideoCamera** - Configuration Information for Video Cameras
- **SMS.VideoServer** - Configuration Information for SMS Video Servers

SMS::abstract Class Reference

Clients should not instantiate this class. Use SMS::SMS_API.

```
#include <SPCommWrapperBase.h>
```

Public Member Functions

void

- **ExecuteManualOverrideTask** (
  int aOverrideTaskID,
  int aDuration,
  String^aOperatorName)
  Execute a Manual Override.

- **ExecuteManualOverrideSet** (
  int aOverrideSetID,
  int aDuration,
  String^aOperatorName)
  Execute a Manual Override set.

- **AcknowlegeAlarm** (
  int aTrnHisID,
  String^aOperatorName)
  Acknowledge an Alarm.

- **SendDatabaseChangeNotice** (unsigned int aDownloadTable)
  Send Database Change Notice to the SP.

- **SendAntipassbackChangeMessage** (
  unsigned int aEncodedID,
  int aState)
  Send Anti-Passback State Change to the SP.

- **RequestDeviceStatus** (
  int aDeviceID,
  int aStatusRequestType,
  int aControllerID)
  Request Device Status.

- **SendTransactionToSP** (
  int aTrnHisID,
  int aTransactionCodeID,
  SYSTEMTIME aTransactionDateTime,
  int aSystemID,
  int aAreaID,
  int aControllerID,
  int aTransDeviceIDactionCodeID,
  int aCardholderID,
  unsigned int aEncodedID,
  int aIssueCode,
  int aCardreadData,
  int aKeyboardData,
  int aFloorNumber,
  int aWorkstationID)

  Sends Transaction to the SP.

virtual void

- **HandleMessageFromSP** (
  int SPMessageType,
  int TrnHisID,
  int TransactionCodeID,
  int AreaID,
  int ControllerID,
  int DeviceID,
  int CardholderID,
  int IssueCode,
  int CardReadData,
  int KeyboardData,
  int FloorNumber,
  int WorkstationID,
  int StatusREquestType,
  unsigned int EncodedID,
  System::DateTime aTransactionDateTime,
  unsigned int aDeviceStatus,
  bool aSecured,
  System::DateTime aSecuredDateTime,
  bool aAcknowledged,
  System::DateTime aAcknowledgedDateTime,
  int aAcknowledgerID,
  int aAlarmPriority,
  int aAlarmLabelID,
  unsigned int aChangedDatabaseTablesBitmap)=0

  DO NOT USE. Use the delegates defined in this class.

- **HandleConnectionStatusChanged** (
  bool aConnectedToSP,
  bool aConnectedToDatabase,
  bool aEncounteredError,
  String^aLastErrorMessage)=0

  DO NOT USE. Instead use the delegates defined in this class.

- **HandleMROCompleted** (
  int aTrnHisID,
  int aStatusCode,
  String^aStatusMessage)=0

  DO NOT USE. Instead use the delegates defined in this class.

Member Function Documentation

void

- **SMS::abstract::AcknowlegeAlarm** (
  int aTrnHisID,
  String^ aOperatorName)
  Acknowledge an Alarm.

- **SMS::abstract::ExecuteManualOverrideSet** (
  int aOverrideSetID,
  int aDuration,
  String^ aOperatorName)
  Execute a Manual Override set.

- **SMS::abstract::ExecuteManualOverrideTask** (
  int aOverrideTaskID,
  int aDuration,
  String^ aOperatorName)
  Execute a Manual Override.

- **SMS::abstract::RequestDeviceStatus** (
  int aDeviceID,
  int aStatusRequestType,
  int aControllerID)
  Request Device Status.

- **SMS::abstract::SendAntipassbackChangeMessage** (
  unsigned int aEncodedID,
  int aState)
  Send Anti-Passback State Change to the SP.

- **SMS::abstract::SendDatabaseChangeNotice** (unsigned int aDownloadTable)
  Send Database Change Notice to the SP.

- **SMS::abstract::SendTransactionToSP** (
  int aTrnHisID,
  int aTransactionCodeID,
  SYSTEMTIME aTransactionDateTime,
  int aSystemID, int aAreaID, int aControllerID,
  int aTransDeviceIDactionCodeID,
  int aCardholderID,
  unsigned int aEncodedID,
  int aIssueCode,
  int aCardreadData,
  int aKeyboardData,
  int aFloorNumber,
  int aWorkstationID)
  Sends Transaction to the SP.

virtual void

- **SMS::abstract::HandleConnectionStatusChanged** (
  bool aConnectedToSP,
  bool aConnectedToDatabase,
  bool aEncounteredError,
  String^ aLastErrorMessage) [pure virtual]
  DO NOT USE. Instead use the delegates defined in this class. Method public due to Win32 DLL Interop requirements.

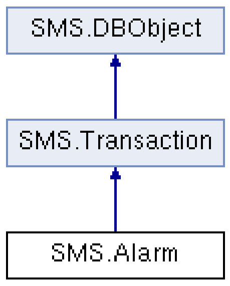
- **SMS::abstract::HandleMessageFromSP** (
  int SPMessageType,
  int TrnHisID,
  int TransactionCodeID,
  int AreaID,
  int ControllerID,
  int DeviceID,
  int CardholderID,
  int IssueCode,
  int CardReadData,
  int KeyboardData,
  int FloorNumber,
  int WorkstationID,
  int StatusRequestType,
  unsigned int EncodedID,
  System::DateTime aTransactionDateTime,
  unsigned int aDeviceStatus, bool aSecured,
  System::DateTime aSecuredDateTime,
  bool aAcknowledged,
  System::DateTime aAcknowledgedDateTime,
  int aAcknowledgerID,
  int aAlarmPriority,
  int aAlarmLabelID,
  unsigned int aChangedDatabaseTablesBitmap) [pure virtual]
  DO NOT USE. Instead use the delegates defined in this class. Method public due to Win32 DLL Interop requirements.

- **SMS::abstract::HandleMROCompleted** (
  int aTrnHisID,
  int aStatusCode,
  String^ aStatusMessage) [pure virtual]
  DO NOT USE. Instead use the delegates defined in this class. Method public due to Win32 DLL Interop requirements.

SMS.Alarm Class Reference

Transaction designated as an **Alarm**; requires **Operator** attention.

Inheritance diagram for SMS.Alarm:



Properties

- bool **isSecured** `[get, set]`

  Alarm has been secured when True.
- DateTime **securedDateTime** `[get, set]`

  Timestamp when Alarm was secured. Ignore when isSecured = False.
- bool **isAcknowledged** `[get, set]`

  Alarm has been acknowledged by an Operator or via this API when True.
- DateTime **acknowledgedDateTime** `[get, set]`

  Timestamp when Alarm was acknowledged. Ignore when isAcknowledged = False.
- string **acknowledgerName** `[get, set]`

  Name of Operator acknowledging the Alarm.
- **Operator acknowledger** `[get, set]`

  Operator acknowledging the Alarm.
- int **alarmCriterialD** `[get, set]`

  Unique ID of the criteria that generated the Alarm.

Property Documentation

- DateTime **SMS.Alarm.acknowledgedDateTime** [get, set]

  Timestamp when Alarm was acknowledged. Ignore when isAcknowledged = False.
- Operator **SMS.Alarm.acknowledger** [get, set]

  Operator acknowledging the Alarm.
  Only populated for Alarms received in real time from the System Processor.
  Not loaded for Alarms retrieved from the database.

- string **SMS.Alarm.acknowledgerName** [get, set]

  Name of SMS Operator acknowledging the Alarm.
  The OperatorID for the Operator acknowledging the Alarm is not stored;
  only the Operator name as a string.

- int **SMS.Alarm.alarmCriteriaID** [get, set]

  Unique ID of the criteria that generated the Alarm.
  SMS.AlarmLabel.alarmLabelID. Populated for Alarms received in real time from the SP.
  Not loaded for logged Alarms retrieved from the database.

- bool **SMS.Alarm.isAcknowledged** [get, set]

  Alarm has been acknowledged by an SMS Operator or via this API when True.

- bool **SMS.Alarm.isSecured** [get, set]

  Alarm has been secured when True.
  Useful for contact Alarms (unsecure until the state of the contact returns to normal) and communication lost Alarms (unsecure until communication is restored). Many types of Transactions that can be designated as Alarms have no notion of securing the Transaction (i.e. an Alarm triggered by an invalid access). In these cases, Alarm.isSecured is always True.

- DateTime **SMS.Alarm.securedDateTime** [get, set]

  Timestamp when Alarm was secured. Ignore when isSecured = False.

SMS.AlarmComment Class Reference

Descriptive notes associated with an **Alarm**.

Properties

- int **alarmCommentID** `[get, set]`

  AlarmComment Unique Identifier.

- int **trnHisID** `[get, set]`

  Unique Transaction ID of the Alarm associated with Comment.

- string **description** `[get, set]`

  Comment description.

- string **operatorName** `[get, set]`

  Name of the Operator entering the Comment.

- DateTime **CommentDateTime** `[get, set]`

  Timestamp when the Comment was generated.
  Local time from Workstation where the Comment was entered.

Property Documentation

- int **SMS.AlarmComment.alarmCommentID** [get, set]

  AlarmComment Unique Identifier.

- DateTime **SMS.AlarmComment.CommentDateTime** [get, set]

  Timestamp when the Comment was generated.
  Local time from Workstation where the Comment was entered.

- string **SMS.AlarmComment.description** [get, set]

  Comment description.

- string **SMS.AlarmComment.operatorName** [get, set]

  Name of the Operator entering the Comment.

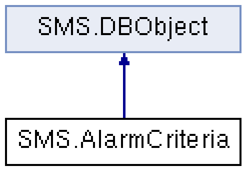
- int **SMS.AlarmComment.trnHisID** [get, set]

  Unique Transaction ID of the Alarm associated with Comment.

SMS.AlarmCriteria Class Reference

Information about the criteria that caused the **Alarm**.

Inheritance diagram for SMS.AlarmCriteria:



Properties

- int **alarmCriterialD** `[get, set]`

  Alarm Criteria Unique Identifier [AlarmLabel.AlarmLabelID].
- DateTime **modifiedDateTime** `[get, set]`

  Alarm Criteria Last Modification Timestamp.
- int **deleteFlag** `[get, set]`

  Alarm Criteria Marked for Deletion.
- bool **rerouteToDefault** `[get, set]`

  Route Alarm Back to Original Destination After Final Destination.
- string **caption** `[get, set]`

  Alarm Criteria Description.
- string **description** `[get, set]`

  Additional optional Alarm Criteria description.
- bool **requiresComments** `[get, set]`

  Operator Must Enter Comments to Acknowledge Alarm.
- bool **forceLogin** `[get, set]`

  Operator Must Login to SMS to Acknowledge Alarm.
- bool **deleteOnReroute** `[get, set]`

  Remove Alarm From Previous Destination on Reroute.
- int **textColor** `[get, set]`

  Alarm Message Text Color.
- int **backgroundColor** `[get, set]`

  Alarm Message Background Color.
- string **fontName** `[get, set]`

  Alarm Message Font Name.

- string **fontStyle** `[get, set]`

  Alarm Message Font Style.
- int **fontSize** `[get, set]`

  Alarm Message Font Size.
- string **UnackUnsecSound** `[get, set]`

  Alarm Unacknowledged and Unsecured Sound Filename.
- string **UnackSecSound** `[get, set]`

  Alarm Unacknowledged and Secured Sound Filename.
- string **AckUnsecSound** `[get, set]`

  Alarm Acknowledged and Unsecured Sound Filename.
- int **UnackUnsecInterval** `[get, set]`

  Alarm Unacknowledged and Unsecured Interval Before Reroute.
- int **UnackSecInterval** `[get, set]`

  Alarm Unacknowledged and Secured Interval Before Reroute.
- int **AckUnsecInterval** `[get, set]`

  Alarm Acknowledged and Unsecured Interval Before Reroute.

Property Documentation

- int **SMS.AlarmCriteria.AckUnsecInterval** `[get, set]`

  Alarm Acknowledged and Unsecured Interval Before Reroute.
- string **SMS.AlarmCriteria.AckUnsecSound** `[get, set]`

  Alarm Acknowledged and Unsecured Sound Filename.
- int **SMS.AlarmCriteria.alarmCriteriaID** `[get, set]`

  Alarm Criteria Unique Identifier [AlarmLabel.AlarmLabelID].
- int **SMS.AlarmCriteria.backgroundColor** `[get, set]`

  Alarm Message Background Color.
- string **SMS.AlarmCriteria.caption** `[get, set]`

  Alarm Criteria Description.
- int **SMS.AlarmCriteria.deleteFlag** `[get, set]`

  Alarm Criteria Marked for Deletion.
- bool **SMS.AlarmCriteria.deleteOnReroute** `[get, set]`

  Remove Alarm From Previous Destination on Reroute.
- string **SMS.AlarmCriteria.description** [get, set]

  Additional optional Alarm Criteria description.
- string **SMS.AlarmCriteria.fontName** [get, set]

  Alarm Message Font Name.
- int **SMS.AlarmCriteria.fontSize** [get, set]

  Alarm Message Font Size.

- string **SMS.AlarmCriteria.fontStyle** [get, set]
  Alarm Message Font Style.
- bool **SMS.AlarmCriteria.forceLogin** [get, set]
  Operator Must Login to SMS to Acknowledge Alarm.
- DateTime **SMS.AlarmCriteria.modifiedDateTime** [get, set]
  Alarm Criteria Last Modification Timestamp.
- bool **SMS.AlarmCriteria.requiresComments** [get, set]
  Operator Must Enter Comments to Acknowledge Alarm.
- bool **SMS.AlarmCriteria.rerouteToDefault** [get, set]
  Route Alarm Back to Original Destination After Final Destination.
- int **SMS.AlarmCriteria.textColor** [get, set]
  Alarm Message Text Color.
- int **SMS.AlarmCriteria.UnackSecInterval** [get, set]
  Alarm Unacknowledged and Secured Interval Before Reroute.
- string **SMS.AlarmCriteria.UnackSecSound** [get, set]
  Alarm Unacknowledged and Secured Sound Filename.
- int **SMS.AlarmCriteria.UnackUnsecInterval** [get, set]
  Alarm Unacknowledged and Unsecured Interval Before Reroute.
- string **SMS.AlarmCriteria.UnackUnsecSound** [get, set]
  Alarm Unacknowledged and Unsecured Sound Filename.

SMS.AlarmInstruction Class Reference

Instructions are associated to the criteria generating an **Alarm**. Alarms generated by the same criteria share Instructions.

Properties

- int **alarmInstructionID** `[get, set]`

  Alarm Instruction Unique Identifier.
- int **alarmCriterialD** `[get, set]`

  Unique ID of AlarmCriteria associated with Alarm Instruction.
- string **description** `[get, set]`

  Alarm Instruction description.

Property Documentation

- int **SMS.AlarmInstruction.alarmCriteriaID** [get, set]

  Unique ID of AlarmCriteria associated with Alarm Instruction.
- int **SMS.AlarmInstruction.alarmInstructionID** [get, set]

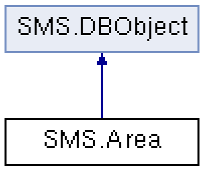
  Alarm Instruction Unique Identifier.
- string **SMS.AlarmInstruction.description** [get, set]

  Alarm Instruction description.

SMS.Area Class Reference

An **Area** is an abstract container that may have zero, one or many devices assigned.

Inheritance diagram for SMS.Area:



Properties

- int **areaID** `[get, set]`

  Area Unique Identifier.

- string **caption** `[get, set]`

  Area Description.

- string **description** `[get, set]`

  Optional additional Area description.

Property Documentation

- int **SMS.Area.areaID** [get, set]

  Area Unique Identifier.

- string **SMS.Area.caption** [get, set]

  Area Description.

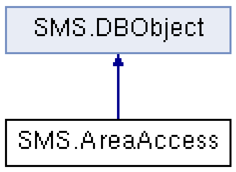
- string **SMS.Area.description** [get, set]

  Optional additional Area description.

SMS.AreaAccess Class Reference

An **AreaAccess** record indicates access to a door.

Inheritance diagram for SMS.AreaAccess:



Properties

- int **areaAccessID** `[get, set]`

  AreaAccess Unique Identifier.
- bool **blocked** `[get, set]`

  Block AreaAccess When True.
- DateTime **activation** `[get, set]`

  The date and time AreaAccess becomes active. Local time at the controller is used, not GMT.
- DateTime **expiration** `[get, set]`

  The date and time AreaAccess ceases to be allowed. Local time at the controller is used, not GMT.
- int **areaID** `[get, set]`

  Unique ID of the Area for the AreaAccess record.
- string **areaCaption** `[get, set]`

  Area description.
- int **cardholderID** `[get, set]`

  Unique ID of the Cardholder for the AreaAccess record.
- string **firstname** `[get, set]`

  Cardholder First Name.
- string **lastname** `[get, set]`

  Cardholder Last Name.
- int **timezoneID** `[get, set]`

  Unique ID of the TimeZone for the AreaAccess record.
- string **timezoneCaption** `[get, set]`

  TimeZone description.

Property Documentation

- DateTime **SMS.AreaAccess.activation** [get, set]

  The date and time AreaAccess becomes active. Local time at the controller is used, not GMT. Controller will apply whichever condition is more restrictive between Cardholder's Activation or AreaAccess Activation in deciding to block or grant Access.

- int **SMS.AreaAccess.areaAccessID** [get, set]

  AreaAccess Unique Identifier.

- string **SMS.AreaAccess.areaCaption** [get, set]

  Area description. Provided for convenience. See Area object.

- int **SMS.AreaAccess.areaID** [get, set]

  Unique ID of the Area for the AreaAccess record.

- bool **SMS.AreaAccess.blocked** [get, set]

  Block AreaAccess When True.
  No physical access will be granted. Provides an easy way for operators to block access without removing the AreaAccess record (for instance, if they want to quickly reactivate it later, without recreating the record).

- int **SMS.AreaAccess.cardholderID** [get, set]

  Unique ID of the Cardholder for the AreaAccess record.

- DateTime **SMS.AreaAccess.expiration** [get, set]

  The date and time AreaAccess ceases to be allowed. Local time at the controller is used, not GMT. Controller will apply whichever condition is more restrictive between Cardholder's Expiration or AreaAccess Expiration in deciding to block or grant Access.

- string **SMS.AreaAccess.firstname** [get, set]

  Cardholder First Name. Provided for convenience. See Cardholder object.

- string **SMS.AreaAccess.lastname** [get, set]

  Cardholder Last Name. Provided for convenience. See Cardholder object.

- string **SMS.AreaAccess.timezoneCaption** [get, set]

  TimeZone description.

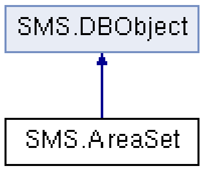
- int **SMS.AreaAccess.timezoneID** [get, set]

  Unique ID of the TimeZone for the AreaAccess record.

SMS.AreaSet Class Reference

An **AreaSet** is an abstract container that may have zero, one or many Areas assigned.

Inheritance diagram for SMS.AreaSet:



Properties

- int **areaSetID** `[get, set]`

  AreaSet Unique Identifier.
- string **caption** `[get, set]`

  AreaSet description.

Property Documentation

- int **SMS.AreaSet.areaSetID** [get, set]

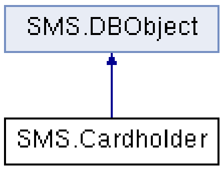
  AreaSet Unique Identifier.
- string **SMS.AreaSet.caption** [get, set]

  AreaSet description.

SMS.Cardholder Class Reference

Cardholders are personnel interacting with card Reader Devices using security Badges.

Inheritance diagram for SMS.Cardholder:



Public Member Functions

- Hashtable **GetUserDefinedFields** ()

  Retrieve User Defined Fields for Cardholder.
- List< Credential > **GetCredentials** ()

  Call SMS_DB.GetCredentials Supplying Only CardholderID.

Properties

- int **cardholderID** `[get, set]`

  Cardholder Unique Identifier.
- string **firstname** `[get, set]`

  Cardholder First Name.
- string **lastname** `[get, set]`

  Cardholder Last Name.
- bool **blocked** `[get, set]`

  Block All Cardholder access when True.
- bool **APControlled** `[get, set]`

  Anti-passback rules enforced for Cardholder at entry Reader Devices when True.
- DateTime **activation** `[get, set]`

  Block All Cardholder access prior to this time.
- DateTime **expiration** `[get, set]`

  Block All Cardholder access after this time.
- Boolean **hasPortrait** `[get]`

  Indicates Cardholder Portrait File available.
- String **portraitPath** `[get]`

  Cardholder Portrait file path.

Member Function Documentation

- List<Credential> SMS.Cardholder.GetCredentials ()

  Call **SMS_DB.GetCredentials** *Supplying Only CardholderID.*

  Returns: List containing Credentials assigned to Cardholder.

- Hashtable **SMS.Cardholder.GetUserDefinedFields** ()

  Retrieve User Defined Fields for Cardholder.
  Column names are the hash keys. UDF values are the hash values.

  Returns: Hash table containing all Cardholder User Defined Fields.

Property Documentation

- DateTime **SMS.Cardholder.activation** [get, set]

  Block All Cardholder access prior to this time.

- bool **SMS.Cardholder.APControlled** [get, set]

  Antipassback rules enforced for Cardholder at entry Reader Devices when True.

- bool **SMS.Cardholder.blocked** [get, set]

  Block All Cardholder access when True.

- int **SMS.Cardholder.cardholderID** [get, set]

  Cardholder Unique Identifier.

- DateTime **SMS.Cardholder.expiration** [get, set]

  Block All Cardholder access after this time.

- string **SMS.Cardholder.firstname** [get, set]

  Cardholder First Name.

- Boolean **SMS.Cardholder.hasPortrait** [get]

  Indicates Cardholder Portrait File available.
  Checks whether a portrait exists by detecting an image file on the file system for this cardholder.
  Warning - a potentially slow operation if the network file access is slow (slower if the path is set incorrectly,
  and the operation times out).

- string **SMS.Cardholder.lastname** [get, set]

  Cardholder Last Name.

- String **SMS.Cardholder.portraitPath** [get]

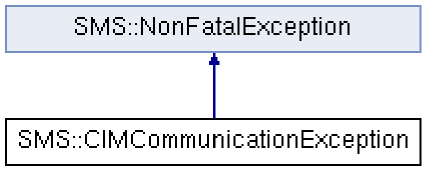
  Cardholder Portrait file path.
  Returns the expected path. Does not validate file existence (use "hasPortrait" or check within client code).

SMS::CIMCommunicationException Class Reference

Exception for signaling CIM communication errors.

`#include <SPCommWrapperBase.h>`

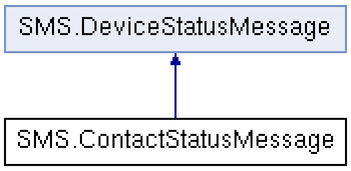
Inheritance diagram for SMS::CIMCommunicationException:



SMS.ContactStatusMessage Class Reference

Contact **Device** Status Message.

Inheritance diagram for SMS.ContactStatusMessage:



Properties

- ContactStatusType **status** [get]

  Contact Status and Type.
- ShuntStatusMessage **reportingShuntStatus** [get]

  Indicates whether reporting of Transactions from this Contact has been shunted, and if so why.
- ShuntStatusMessage **triggersShuntStatus** [get]

  Indicates whether Device Triggers from this Contact have been shunted, and if so why.
- ShuntStatusMessage **faultReportingShuntStatus** [get]

  Indicates whether Fault Reporting from this Contact has been shunted, and if so why.
- ShuntStatusMessage **faultTriggersShuntStatus** [get]

  Indicates whether Fault Triggers from this Contact have been shunted, and if so why.

Property Documentation

- ShuntStatusMessage **SMS.ContactStatusMessage.faultReportingShuntStatus** [get]

  Indicates whether Fault Reporting from this Contact has been shunted, and if so why.
- ShuntStatusMessage **SMS.ContactStatusMessage.faultTriggersShuntStatus** [get]

  Indicates whether Fault Triggers from this Contact have been shunted, and if so why.
- ShuntStatusMessage **SMS.ContactStatusMessage.reportingShuntStatus** [get]

  Indicates whether reporting of Transactions from this Contact has been shunted, and if so why.
- ContactStatusType **SMS.ContactStatusMessage.status** [get]

  Contact Status and Type.
  Will indicate if the Contact is compromised (shorted or open) for supervised contacts.
- ShuntStatusMessage **SMS.ContactStatusMessage.triggersShuntStatus** [get]

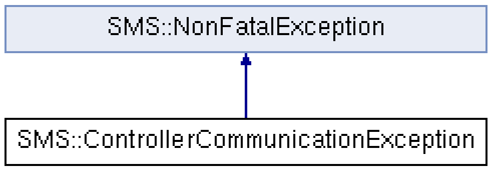
  Indicates whether Device Triggers from this Contact have been shunted, and if so why.

SMS::ControllerCommunicationException Class Reference

Exception for signaling Controller communication errors.

```
#include <SPCommWrapperBase.h>
```

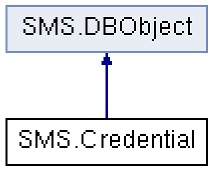
Inheritance diagram for SMS::ControllerCommunicationException:



SMS.Credential Class Reference

Credentials are items (typically badges) presented to a Reader to gain access.

Inheritance diagram for SMS.Credential:



Properties

- int **badgeID** `[get, set]`

  Unique Badge identifier.

- string **encodedID** `[get, set]`

  The ID information physically encoded on the badge. Unique among all other active Credentials; typically an integer value.

- int **stampedID** `[get, set]`

  Numeric value printed on the Credential (not data encoded within the Credential read by the Reader). Set to zero by default.

- int **keypadID** `[get, set]`

  PIN used in combination with Credential.

- int **badgeTechnologyID** `[get, set]`

  Unique ID of Credential technology (i.e. proximity, magstripe, etc).

- string **badgeTechnologyCaption** `[get, set]`

  Credential technology description.

- int **cardholderID** `[get, set]`

  Unique ID of Cardholder assigned to Credential.

Property Documentation

- int **SMS.Credential.badgeID** [get, set]

  Unique Badge identifier.
  A database generated incrementing ID.
  Has no direct relationship to data physically encoded on the credential.

- string **SMS.Credential.badgeTechnologyCaption** [get, set]

  Credential technology description.

- int **SMS.Credential.badgeTechnologyID** [get, set]

  Unique ID of Credential technology (i.e. proximity, magstripe, etc).

- int **SMS.Credential.cardholderID** [get, set]

  Unique ID of Cardholder assigned to Credential.

- string **SMS.Credential.encodedID** [get, set]

  The ID information physically encoded on the badge.
  Unique among all other active Credentials; typically an integer value.
  Stored as a string in the database.

- int **SMS.Credential.keypadID** [get, set]

  PIN used in combination with Credential.
  Set to zero when no PIN used with Credential.

- int **SMS.Credential.stampedID** [get, set]

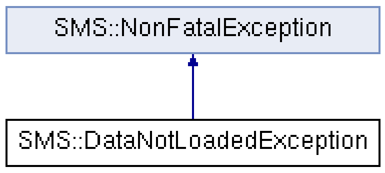
  Numeric value printed on the Credential (not data encoded within the Credential read by the Reader).
  Set to zero by default.

SMS::DataNotLoadedException Class Reference

Exception for signaling that internal reference data required by **SMS_API** was not successfully loaded.

```
#include <SPCommWrapperBase.h>
```

Inheritance diagram for SMS::DataNotLoadedException:

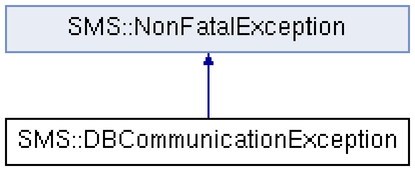


SMS::DBCommunicationException Class Reference

Exception for signaling database communication errors.

```
#include <SPCommWrapperBase.h>
```

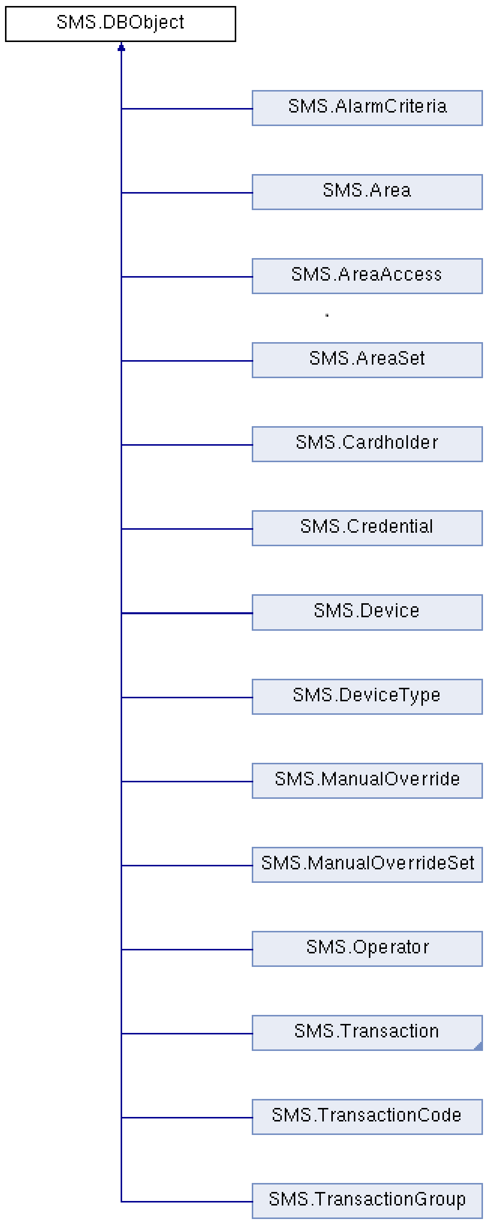
Inheritance diagram for SMS::DBCommunicationException:



SMS.DBObject Class Reference

SMS_API objects with information stored in the SMS database inherit from this abstract class which provides the foundations for basic database I/O.

Inheritance diagram for SMS.DBObject:



Public Member Functions

- **DBObject** (**SMS_API** aAPI, SqlDataReader aOpenQuery=null)

  SchalgeAPI objects with information stored in the SMS database inherit from this object, which provides the foundations for basic database I/O.

- void **Initialize** (int aUniqueID)

  Loads Object from the database using the unique ID provided.

- void **Initialize** ()

  Base DBObject initialization method that uses the uniqueID property of the Object. Commonly used for objects received via real time communication.

Constructor & Destructor Documentation

SMS.DBObject.DBObject (SMS_API **aAPI,** SqlDataReader **aOpenQuery** = null)

SMS_API objects with information stored in the SMS database inherit from this object, which provides the foundations for basic database I/O.

Parameters:

aAPI	*SMS_API Instance.*
aOpenQuery	*Optional Data Reader Object.*

Member Function Documentation

void **SMS.DBObject.Initialize** (int **aUniqueID)**

Loads Object from the database using the unique ID provided.

Parameters:

- aUniqueID *Initialize Object with UniqueID*

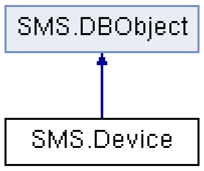
void **SMS.DBObject.Initialize** ()

Base DBObject initialization method that uses the uniqueID property of the Object.
Commonly used for objects received via real time communication.
Do Not Call if uniqueID has not been set. Behavior is undefined and will likely lead to an exception.

SMS.Device Class Reference

Devices are things like Readers, Contacts, Workstations, etc.

Inheritance diagram for SMS.Device:



Properties

- int **deviceID** `[get, set]`

  Device Unique Identifier.
- String **caption** `[get, set]`

  Device description.
- int **deviceTypeID** `[get, set]`

  Unique ID of DeviceType.
- String **deviceTypeCaption** `[get, set]`

  DeviceType description.
- int **areaID** `[get, set]`

  Unique ID of Area Device is assigned.
- int **parentID** `[get, set]`

  Unique ID of Parent Device (i.e. the Parent controller for a RINX or the RINX for most DODs).
- int **contactTypeID** `[get, set]`

  Unique Identifier for ContactType (if Device is a Contact).
- string **contactTypeCaption** `[get, set]`

  ContactType description (if appropriate).
- DateTime **modifiedDateTime** `[get, set]`

  Timestamp for last Device modification (in GMT).

Property Documentation

- int **SMS.Device.areaID** [get, set]

  Unique ID of Area Device is assigned.
- String **SMS.Device.caption** [get, set]

  Device description.

- string **SMS.Device.contactTypeCaption** [get, set]

  ContactType description (if appropriate).
- int **SMS.Device.contactTypeID** [get, set]

  Unique Identifier for ContactType (if Device is a Contact).
- int **SMS.Device.deviceID** [get, set]

  Device Unique Identifier.
- String **SMS.Device.deviceTypeCaption** [get, set]

  DeviceType description.
- int **SMS.Device.deviceTypeID** [get, set]

  Unique ID of DeviceType.
- DateTime **SMS.Device.modifiedDateTime** [get, set]

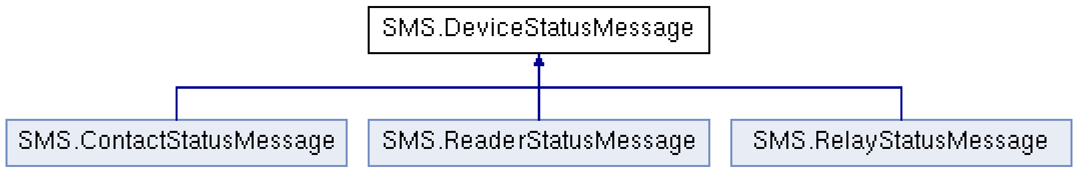
  Timestamp for last Device modification (in GMT).
  Only initialized by calls to GetDevice() or Device.Initialize().
  Device objects returned as part of Transaction.Initialize() or Alarm.Initialize(),
  do not load the value of this information.
- int **SMS.Device.parentID** [get, set]

  Unique ID of Parent Device (i.e. the Parent controller for a RINX or the RINX for most DODs).
  Set to 0 for Devices without a Parent such as a Workstation.
  Set to -1 for the special "All Devices" record (self referential).

SMS.DeviceStatusMessage Class Reference

Device Status Message.

Inheritance diagram for SMS.DeviceStatusMessage:



Properties

- int **deviceID** `[get, set]`

    Device ID of the Device Status.

- **StatusRequestType type** `[get, set]`

    Type of Status (Reader, Relay, or Contact).

- uint **statusBitmap** `[get, set]`

    Status Message raw data.

Property Documentation

- int **SMS.DeviceStatusMessage.deviceID** [get, set]

    Device ID of the Device Status.

- uint **SMS.DeviceStatusMessage.statusBitmap** [get, set]

    Status Message raw data. Used for debugging. Client code should ignore.

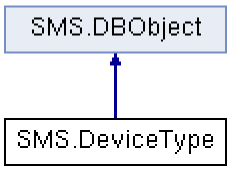
- StatusRequestType **SMS.DeviceStatusMessage.type** [get, set]

    Type of Status (Reader, Relay, or Contact).

SMS.DeviceType Class Reference

Indicates the types of devices that can exist (Readers, Contacts, Workstations, etc).

Inheritance diagram for SMS.DeviceType:



Properties

- int **deviceTypeID** `[get, set]`

    DeviceType Unique Identifier.

- string **caption** `[get, set]`

    DeviceType description.

Property Documentation

- string **SMS.DeviceType.caption** [get, set]

    DeviceType description.

- int **SMS.DeviceType.deviceTypeID** [get, set]

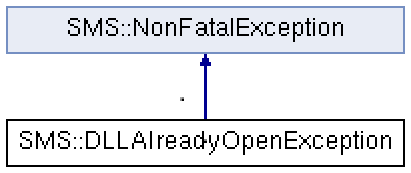
    DeviceType Unique Identifier.

SMS::DLLAlreadyOpenException Class Reference

Exception for signaling SMS_API.dll already open errors.

`#include <SPCommWrapperBase.h>`

Inheritance diagram for SMS::DLLAlreadyOpenException:

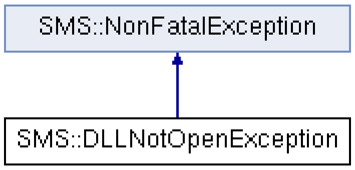


SMS::DLLNotOpenException Class Reference

Exception for signaling SchlagAPI.dll not open errors.

`#include <SPCommWrapperBase.h>`

Inheritance diagram for SMS::DLLNotOpenException:



SMS::FatalException Class Reference

Exception for signaling fatal errors.

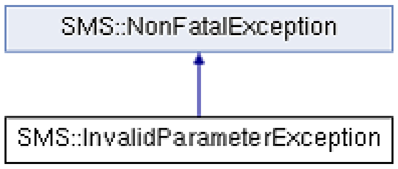
```
#include <SPCommWrapperBase.h>
```

SMS::InvalidParameterException Class Reference

Exception for signaling that one or more parameters passed to a routine are invalid.

```
#include <SPCommWrapperBase.h>
```

Inheritance diagram for SMS::InvalidParameterException:

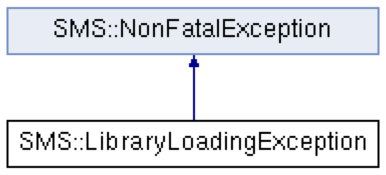


SMS::LibraryLoadingException Class Reference

Exception for signaling library loading errors.

```
#include <SPCommWrapperBase.h>
```

Inheritance diagram for SMS::LibraryLoadingException:

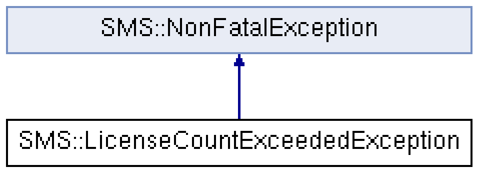


SMS::LicenseCountExceededException Class Reference

Exception for signaling that **SMS_API** is licensed, but the total number of workstation and/or API logins exceeds the number of licensed client logins.

```
#include <SPCommWrapperBase.h>
```

Inheritance diagram for SMS::LicenseCountExceededException:

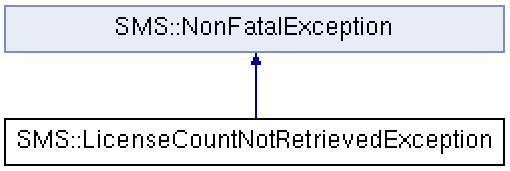


SMS::LicenseCountNotRetrievedException Class Reference

Exception for signaling that an error occurred retrieving license count information from the SP.

```
#include <SPCommWrapperBase.h>
```

Inheritance diagram for SMS::LicenseCountNotRetrievedException:

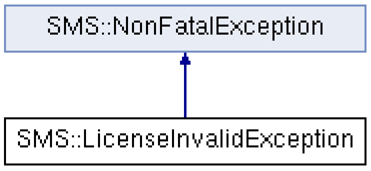


SMS::LicenseInvalidException Class Reference

Exception for signaling that **SMS_API** is not a licensed feature for this SMS installation.

```
#include <SPCommWrapperBase.h>
```

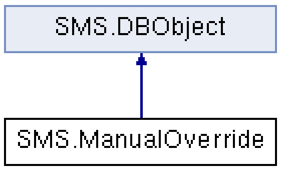
Inheritance diagram for SMS::LicenseInvalidException:



SMS.ManualOverride Class Reference

A **ManualOverride** (MRO) is used to initiate a **Device** action (e.g. open a door, energize a relay, suspend contact reporting, etc).

Inheritance diagram for SMS.ManualOverride:



Properties

- int **manualOverrideID** `[get, set]`

  MRO Unique Identifier.
- String **caption** `[get, set]`

  MRO description.
- String **description** `[get, set]`

  MRO optional additional description.
- int **deviceID** `[get, set]`

  Unique Identifier for the Device associated with MRO.
- string **deviceCaption** `[get, set]`

  Device description.
- DateTime **modifiedDateTime** `[get, set]`

  Last MRO modification timestamp (in GMT).

Property Documentation

- String **SMS.ManualOverride.caption** [get, set]

  MRO description.
- String **SMS.ManualOverride.description** [get, set]

  MRO optional additional description.
- string **SMS.ManualOverride.deviceCaption** [get, set]

  Device description.
- int **SMS.ManualOverride.deviceID** [get, set]

  Unique Identifier for the Device associated with MRO.
- int **SMS.ManualOverride.manualOverrideID** [get, set]

  MRO Unique Identifier.

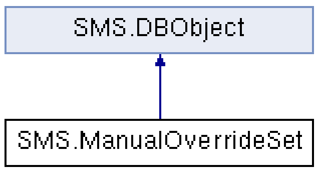
- DateTime **SMS.ManualOverride.modifiedDateTime** [get, set]

  Last MRO modification timestamp (in GMT).

SMS.ManualOverrideSet Class Reference

A **ManualOverrideSet** is a collection of MROs.

Inheritance diagram for SMS.ManualOverrideSet:



Properties

- int **manualOverrideSetID** `[get, set]`

  MRO Set Unique Identifier.
- String **caption** `[get, set]`

  MRO Set description.
- String **description** `[get, set]`

  MRO Set optional additional description.

Property Documentation

- String **SMS.ManualOverrideSet.caption** [get, set]

  MRO Set description.
- String **SMS.ManualOverrideSet.description** [get, set]

  MRO Set optional additional description.
- int **SMS.ManualOverrideSet.manualOverrideSetID** [get, set]

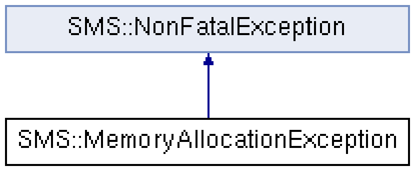
  MRO Set Unique Identifier.

SMS::MemoryAllocationException Class Reference

Exception for signaling that memory could not be allocated on the global heap.

```
#include <SPCommWrapperBase.h>
```

Inheritance diagram for SMS::MemoryAllocationException:

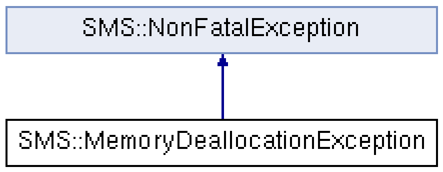


SMS::MemoryDeallocationException Class Reference

Exception for signaling that memory could not be deallocated on the global heap.

```
#include <SPCommWrapperBase.h>
```

Inheritance diagram for SMS::MemoryDeallocationException:

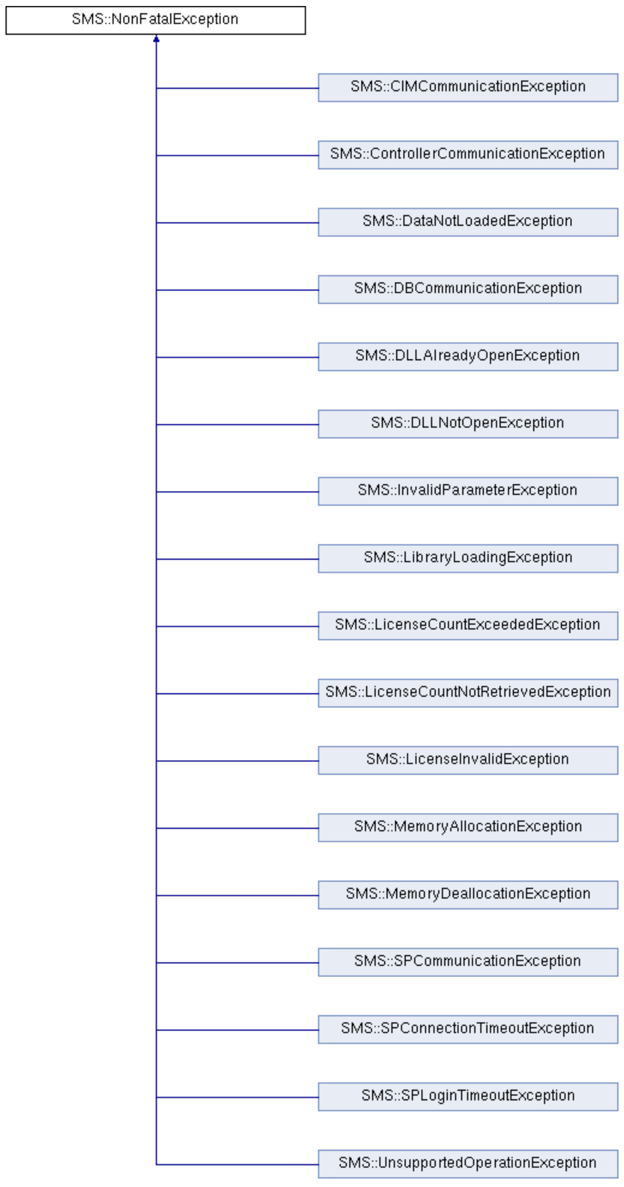


SMS::NonFatalException Class Reference

Exception for signaling non fatal errors.

```
#include <SPCommWrapperBase.h>
```

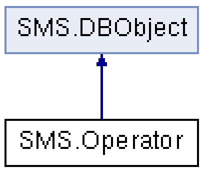
Inheritance diagram for SMS::NonFatalException:


SMS::NonFatalException
SMS::CIMCommunicationException
SMS::ControllerCommunicationException
SMS::DataNotLoadedException
SMS::DBCommunicationException
SMS::DLLAlreadyOpenException
SMS::DLLNotOpenException
SMS::InvalidParameterException
SMS::LibraryLoadingException
SMS::LicenseCountExceededException
SMS::LicenseCountNotRetrievedException
SMS::LicenseInvalidException
SMS::MemoryAllocationException
SMS::MemoryDeallocationException
SMS::SPCommunicationException
SMS::SPConnectionTimeoutException
SMS::SPLoginTimeoutException
SMS::UnsupportedOperationException

SMS.Operator Class Reference

SMS application **Operator**.

Inheritance diagram for SMS.Operator:



Properties

- int **operatorID** `[get, set]`

  Operator Unique Identifier.
- String **initials** `[get, set]`

  Unique login name string used by the Operator.
- string **firstname** `[get, set]`

  Operator's first name.
- string **lastname** `[get, set]`

  Operator's last name.

Property Documentation

- string **SMS.Operator.firstname** [get, set]

  Operator's first name.
- String **SMS.Operator.initials** [get, set]

  Unique login name string used by the Operator.
- string **SMS.Operator.lastname** [get, set]

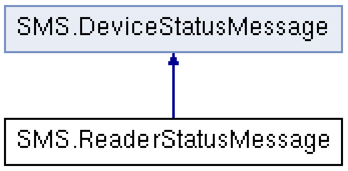
  Operator's last name.
- int **SMS.Operator.operatorID** [get, set]

  Operator Unique Identifier.

SMS.ReaderStatusMessage Class Reference

Reader **Device** Status Message.

Inheritance diagram for SMS.ReaderStatusMessage:



Properties

- bool **communicating** [get]

  Indicates Reader is communicating with the controller.
- bool **keypadEnabled** [get]

  Indicated whether a Keypad is Present AND Enabled.
- ShuntStatusMessage **reportingShuntStatus** [get]

  Indicates whether reporting of Transactions from this Reader has been shunted, and if so why.
- ShuntStatusMessage **triggersShuntStatus** [get]

  Indicates whether Device Triggers from this Reader have been shunted, and if so why.

Property Documentation

- bool **SMS.ReaderStatusMessage.communicating** [get]

  Indicates Reader is communicating with the controller.
- bool **SMS.ReaderStatusMessage.keypadEnabled** [get]

  Indicated whether a Keypad is Present AND Enabled.
- ShuntStatusMessage **SMS.ReaderStatusMessage.reportingShuntStatus** [get]

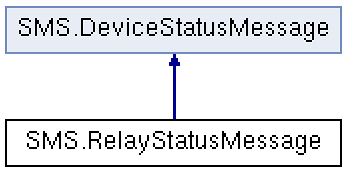
  Indicates whether reporting of Transactions from this Reader has been shunted, and if so why.
- ShuntStatusMessage **SMS.ReaderStatusMessage.triggersShuntStatus** [get]

  Indicates whether Device Triggers from this Reader have been shunted, and if so why.

SMS.RelayStatusMessage Class Reference

Relay **Device** Status Message.

Inheritance diagram for SMS.RelayStatusMessage:



Properties

- RelayStatusType **status** [get]

  Indicates if the Relay is Energized or Released.
- StatusReason **statusReason** [get]

  Indicates the reason Relay is Energized or Released.
- StatusDuration **statusDuration** [get]

  Indicates the Relay Status Duration Type. Not Recommended for Use.
- ShuntStatusMessage **reportingShuntStatus** [get]

  Indicates whether reporting of Transactions from this Relay has been shunted, and if so why.
- ShuntStatusMessage **triggersShuntStatus** [get]

  Indicates whether Device Triggers from this Relay have been shunted, and if so why.

Property Documentation

- ShuntStatusMessage **SMS.RelayStatusMessage.reportingShuntStatus** [get]

  Indicates whether reporting of Transactions from this Relay has been shunted, and if so why.
- RelayStatusType **SMS.RelayStatusMessage.status** [get]

  Indicates if the Relay is Energized or Released.
- StatusDuration **SMS.RelayStatusMessage.statusDuration** [get]

  Indicates the Relay Status Duration Type. Not Recommended for Use.
- StatusReason **SMS.RelayStatusMessage.statusReason** [get]

  Indicates the reason Relay is Energized or Released.
- ShuntStatusMessage **SMS.RelayStatusMessage.triggersShuntStatus** [get]

  Indicates whether Device Triggers from this Relay have been shunted, and if so why.

SMS.SMS_API Class Reference

Primary Class for SMS Access Control System Interaction. Documentation for many important methods are found in parent class SPCommWrapperBase.

Public Member Functions

- void **RequestStatus** (int **aDeviceID)**

  Request Status Message from System Processor ASAP for Device.

- void **ResetAntipassbackStatus** (uint **aEncodedID)**

  Reset Antipassback State for Credential.

- override void **HandleMessageFromSP** (
  int **SPMessageType**,
  int **TrnHisID**,
  int **TransactionCodeID**,
  int **AreaID**,
  int **ControllerID**,
  int **DeviceID**,
  int **CardholderID**,
  int **IssueCode**,
  int **CardReadData**,
  int **KeyboardData**,
  int **FloorNumber**,
  int **WorkstationID**,
  int **StatusRequestType**,
  uint **EncodedID**,
  DateTime **aTransactionDateTime**,
  uint **aDeviceStatus**,
  bool **aSecured**,
  DateTime **aSecuredDateTime**,
  bool **aAcknowledged**,
  DateTime **aAcknowledgedDateTime**,
  int **aAcknowledgerID**,
  int **aAlarmPriority**,
  int **aAlarmLabelID**,
  uint **aChangedDatabaseTablesBitmap**)

  Handle Message from SP = DO NOT USE.

- override void **HandleConnectionStatusChanged** (
  bool **aConnectedToSP**,
  bool **aConnectedToDatabase**,
  bool **aEncounteredError**,
  String **aLastErrorMessage**)

  Handle Connection Status Changed = DO NOT USE.

- override void **HandleMROCompleted** (
  int **aTrnHisID**,
  int **aLastStatusCode**,
  String **aLastStatusMessage**)

  Handle MRO Completed = DO NOT USE.

- String **ReturnCodeToString** (int **aReturnCode)**

  Method for coverting an SMS_API ReturnCode to a string.

Public Attributes

- TransactionHandler **transactionHandler**

  Transaction handler.
- AlarmHandler **alarmHandler**

  Alarm handler.
- AlarmAcknowledgementHandler **alarmAcknowledgementHandler**

  Alarm acknowledgement handler.
- AlarmKillHandler **alarmKillHandler**

  The Alarm kill handler.
- DatabaseChangeHandler **databaseChangeHandler**

  Database change handler.
- DeviceStatusChangeHandler **deviceStatusChangeHandler**

  Device status change handler.
- MROExecutionCompleteHandler **mROExecutionCompleteHandler**

  MRO execution complete handler.
- ConnectionStatusChangedHandler **connectionStatusChangedHandler**

  Connection status changed handler.

Properties

- String **DataDirectoryPath** [get, set]

  SMS Data Directory Path.
- SMS_DB **DB** [get]

  SMS Database Object.

Member Function Documentation

override void **SMS.SMS_API.HandleConnectionStatusChanged** (
bool **aConnectedToSP**,
bool **aConnectedToDatabase**,
bool **aEncounteredError**,
String **aLastErrorMessage**)

Handle Connection Status Changed = DO NOT USE.
Clients should never use this method. Use the delegates defined in this class.
Public method required to support Win32 DLL Interop.

override void **SMS.SMS_API.HandleMessageFromSP** (
int **SPMessageType**,
int **TrnHisID**,
int **TransactionCodeID**,
int **AreaID**,
int **ControllerID**,
int **DeviceID**,
int **CardholderID**,
int **IssueCode**,
int **CardReadData**,
int **KeyboardData**,
int **FloorNumber**,
int **WorkstationID**,
int **StatusRequestType**,
uint **EncodedID**,
DateTime **aTransactionDateTime**,
uint **aDeviceStatus**,
bool **aSecured**,
DateTime **aSecuredDateTime**,
bool **aAcknowledged**,
DateTime **aAcknowledgedDateTime**,
int **aAcknowledgerID**,
int **aAlarmPriority**,
int **aAlarmLabelID**,
uint **aChangedDatabaseTablesBitmap**)

Handle Message from SP = DO NOT USE.
Clients should never use this method. Use the delegates defined in this class.
Public method required to support Win32 DLL Interop.

override void **SMS.SMS_API.HandleMROCompleted** (
int **aTrnHisID**,
int **aLastStatusCode**,
String **aLastStatusMessage**)

Handle MRO Completed = DO NOT USE.
Clients should never use this method. Use the delegates defined in this class.
Public method required to support Win32 DLL Interop.

void **SMS.SMS_API.RequestStatus** (int **aDeviceID**)

Request Status Message from System Processor ASAP for Device.
Reader, Relay and Contact Status is supported. Clients should expect DeviceStatus handler delegate will be invoked if the device is currently communicating.

Parameters:

- aDeviceID *Return Status for DeviceID*

void **SMS.SMS_API.ResetAntipassbackStatus** (uint **aEncodedID**)

Reset Antipassback State for Credential.

Parameters:

- aDeviceID *Return Status for DeviceID*

String **SMS.SMS_API.ReturnCodeToString** (int **aReturnCode**)

Method for converting an SMS_API ReturnCode to a string.

Parameters:

- aReturnCode *SMS_API initialization Return Code*

Member Data Documentation

- AlarmAcknowledgementHandler **SMS.SMS_API.alarmAcknowledgementHandler**

  Alarm acknowledgement handler. See SMS.AlarmAcknowledgementHandler (delegate).
- AlarmHandler **SMS.SMS_API.alarmHandler**

  Alarm handler. See SMS.AlarmHandler (delegate).
- AlarmKillHandler **SMS.SMS_API.alarmKillHandler**

  The Alarm kill handler. See SMS.AlarmKillHandler (delegate).
- ConnectionStatusChangedHandler **SMS.SMS_API.connectionStatusChangedHandler**

  Connection status changed handler. See SMS.ConnectionStatusChangedHandler (delegate).
- DatabaseChangeHandler **SMS.SMS_API.databaseChangeHandler**

  Database change handler. See SMS.DatabaseChangeHandler (delegate).
- DeviceStatusChangeHandler **SMS.SMS_API.deviceStatusChangeHandler**

  Device status change handler. See SMS.DeviceStatusChangeHandler (delegate).
- MROExecutionCompleteHandler **SMS.SMS_API.mROExecutionCompleteHandler**

  MRO execution complete handler. See SMS.MROExecutionCompleteHandler (delegate).
- TransactionHandler **SMS.SMS_API.transactionHandler**

  Transaction handler. See SMS.TransactionHandler (delegate).

Property Documentation

- String **SMS.SMS_API.DataDirectoryPath** [get, set]

  SMS Data Directory Path.
- SMS_DB **SMS.SMS_API.DB** [get]

  SMS Database Object.

SMS.SMS_DB Class Reference

This class abstracts database interaction and contains methods that return lists of DBObjects.

Public Member Functions

- Hashtable **GetUserDefinedFields** (int **aCardholderID)**

  Retrieve All User Defined Field Values for Cardholder.

- List< TransactionCode > **GetTransactionCodes** (
  int?**aTransactionCodeID=**null,
  int?**aTransactionCodeHi**=null)

  Retrieve Transaction Codes.

- List< TransactionGroup > **GetTransactionGroups** (int?**aTransactionCodeHi**=null)

  Retrieve Transaction Groups.

- List< Credential > **GetCredentials** (
  int?**aBadgeID**=null,
  string **aEncodedID**=null,
  int?**aStampedID**=null,
  int?**aCardholderID**=null)

  Retrieve Credentials.

- List< Area > **GetAreas** (
  int?**aAreaID**=null,
  string **aCaptionPrefix**=null,
  int?**aAreaSetID**=null)

  Retrieve Areas.

- List< AreaSet > **GetAreaSets** (
  int?**aAreaSetID**=null,
  string **aCaptionPrefix**=null)

  Retrieve AreasSets.

- List< AreaAccess > **GetAreaAccesses** (
  int?**aAreaAccessID**=null,
  int?**aAreaID**=null,
  int?**aCardholderID**=null)

  Retrieve Area Accesses.

- List< Cardholder > **GetCardholders** (
  int?**aCardholderID**=null,
  string **aFirstnamePrefix**=null,
  string **aLastnamePrefix**=null,
  string **aBadgeEncodedID**=null,
  bool?**aBlocked**=null,
  int?**aAreaID**=null)

  Retrieve Cardholders

  .

- List< DeviceType > **GetDeviceTypes** ()

  Retrieve Device Types.

- List< Device > **GetDevices** (
  int?**aDeviceID**=null,
  string **aCaptionPrefix**=null,
  int?**aAreaID**=null,
  int?**aDeviceTypeID**=null,
  DateTime?**aModifiedDateTime**=null)

  Retrieve Devices.

- List< Device > **GetDeletedDevices** (
  int?**aDeviceID**=null,
  string **aCaptionPrefix**=null,
  int?**aAreaID**=null,
  int?**aDeviceTypeID**=null,
  DateTime?**aDeleteStart**=null,
  DateTime?**aDeleteEnd**=null)

  Retrieve Deleted Devices.

- List< Transaction > **GetTransactions** (
  DateTime?**aStartTime**=null,
  DateTime?**aEndTime**=null,
  int?**aCardholderID**=null,
  int?**aAreaID**=null,
  int?**aDeviceID**=null)

  Retrieve Logged Transactions.

- List< Alarm > **GetAlarms** (
  DateTime?**aStartTime**=null,
  DateTime?**aEndTime**=null,
  int?**aCardholderID=**null,
  int?**aAreaID**=null,
  int?**aDeviceID**=null)

  Retrieve Logged Alarms.

- List< AlarmComment > **GetAlarmComments** (int **aTrnHisID**)

  Retrieve Alarm Comments.

- bool InsertAlarmComment (
  int **aTrnHisID**,
  string **aComment**,
  string **aOperatorName**,
  DateTime **aCommentDateTime**)

  Insert an Alarm Comment.

- List< AlarmInstruction > GetAlarmInstructions (int **aAlarmCriteriaID**)

  Retrieve Alarm Instructions.

- List< ManualOverride > GetManualOverrides (
  int?**aDeviceID**=null,
  DateTime?**aModifiedDateTime**=null)

  Retrieve Manual Overrides (MROs).

- List< ManualOverrideSet > GetManualOverrideSets ()

  Retrieve ManualOverrideSets (MRO Sets).

- List< VideoServer > GetVideoServers ()

  Retrieve Enabled Video Servers.

- List< VideoCamera > GetVideoCameras (
  int?**VideoServerID**=null,
  int?**DeviceID**=null,
  int?**TranCodeHi**=null,
  int?**TranCodeLo**=null,
  int?**TranCodeID**=null)

  Retrieve Configured Camera Control Entries.

Properties

- SqlConnection **SQLConnection** `[get]`

  Connection object which may be used to run SQL queries directly.

Member Function Documentation

List<AlarmComment> **SMS.SMS_DB.GetAlarmComments** (int **aTrnHisID**)

Retrieve Alarm Comments.

Parameters:

- aTrmHisID *Return Alarm Comments with TrnHisID.*

Returns: List Containing Alarm Comments.

List<AlarmInstruction> **SMS.SMS_DB.GetAlarmInstructions** (int **aAlarmCriteriaID**)

Retrieve Alarm Instructions.

Parameters:

▪ aAlarmCriteria	*AlarmCriteriaID (alarmLabelID).*

Returns: List Containing Alarm Instructions.

List<Alarm> **SMS.SMS_DB.GetAlarms** (
DateTime? **aStartTime** = null,
DateTime? **aEndTime** = null,
int? **aCardholderID** = null,
int? **aAreaID** = null,
int? **aDeviceID** = null)

Retrieve Logged Alarms.

Parameters:

▪ aStartTime	*Optional Start Time.*
▪ aEndTime	*Optional End Time.*
▪ aCardholderID	*Optional CardholderID.*
▪ aAreaID	*Optional AreaID.*
▪ aDeviceID	*Optional DeviceID.*

Returns: List containing Alarms.

List<AreaAccess> **SMS.SMS_DB.GetAreaAccesses** (
int? **aAreaAccessID** = null, int?
aAreaID = null,
int? **aCardholderID** = null)

Retrieve Area Accesses.

Parameters:

▪ aAreaAccessID	*Optional AreaAccessID.*
▪ aAreaID	*Optional AreaID.*
▪ aCardholderID	*Optional CardholderID.*

Returns: List containing Area Accesses.

List<Area> **SMS.SMS_DB.GetAreas** (
int? **aAreaID** = null,
string **aCaptionPrefix** = null,
int? **aAreaSetID** = null)

Retrieve Areas.

Parameters:

▪ aAreaID	*Optional AreaID.*
▪ aCaptionPrefix	*Optional CaptionPrefix.*
▪ aAreaSetID	*Optional AreaSetID.*

Partial Match on CaptionPrefix

Returns: List containing Areas.

List<AreaSet> SMS.SMS_DB.GetAreaSets (
int? **aAreaSetID** = null,
string **aCaptionPrefix** = null)

Retrieve AreasSets.

Parameters:

▪ aAreaSetID	*Optional AreaSetID.*
▪ aCaptionPrefix	*Optional CaptionPrefix.*

Partial Match on CaptionPrefix.

Returns: List containing Area Sets.

List<Cardholder> **SMS.SMS_DB.GetCardholders** (
int? **aCardholderID** = null,
string **aFirstnamePrefix** = null,
string **aLastnamePrefix** = null,
string **aBadgeEncodedID** = null,
bool? **aBlocked** = null,
int? **aAreaID** = null)

Retrieve Cardholders.
Parameters may be left Null (to return all cardholders) or "mixed and matched" (i.e. to return only blocked cardholders whose last name is like "Smith").

Parameters:

▪ aCardholderID	*Optional CardholderID.*
▪ aFirstnamePrefix	*Optional FirstnamePrefix.*
▪ aLastnamePrefix	*Optional LastnamePrefix.*
▪ aBadgeEncodedID	*Optional BadgeEncodedID.*

▪ aBlocked	*Optional Blocked = True or False.*
▪ aAreaID	*Optional AreaID.*

Partial Matches on FirstnamePrefix and LastnamePrefix.

Returns: List containing Cardholders.

List<Credential> **SMS.SMS_DB.GetCredentials** (
int? **aBadgeID** = null,
string **aEncodedID** = null,
int? **aStampedID** = null,
int? **aCardholderID** = null)

Retrieve Credentials.

Parameters:

▪ aBadgeID	*Optional BadgeID.*
▪ aEncodedID	*Optional EncodedID.*
▪ aStampedID	*Optional StampedID.*
▪ aCardholderID	*Optional CardholderID.*

Returns: List containing Credentials.

List<Device> **SMS.SMS_DB.GetDeletedDevices** (
int? **aDeviceID** = null,
string **aCaptionPrefix** = null,
int? **aAreaID** = null,
int? **aDeviceTypeID** = null,
DateTime? **aDeleteStart** = null,
DateTime? **aDeleteEnd** = null)

Retrieve Deleted Devices.

Parameters:

▪ aDeviceID	*Optional DeviceID.*
▪ aAreaID	*Optional AreaID.*
▪ aCaptionPrefix	*Optional CaptionPrefix.*
▪ aDeviceTypeID	*Optional DeviceTypeID.*
▪ aDeleteStart	*Optional DeleteStart.*
▪ aDeleteEnd	*Optional DeleteEnd.*

Returns: List containing Deleted Devices.

A Device deleted by the end user is not initially deleted from the database. The device has a DeleteFlag set and the ModifiedDateTime field updated. This routine will retrieve all Devices with the DeleteFlag set and also meeting any of the optional criteria supplied. Once the Database Maintenance utility is run and the Device is removed from the SMS database, it cannot be retrieved. Therefore, it is recommended that the list of deleted Devices be retrieved and cached locally using a narrow date range as soon HandleDatabaseChange EventArgs indicate a possible deletion. Partial matches on CaptionPrefix are performed. End Date is ignored unless Start Date is also provided.

List<Device> **SMS.SMS_DB.GetDevices** (
int? **aDeviceID** = null,
string **aCaptionPrefix** = null,
int? **aAreaID** = null,
int? **aDeviceTypeID** = null,
DateTime? **aModifiedDateTime** = null)

Retrieve Devices.

Parameters:

aDeviceID	*Optional DeviceID.*
aAreaID	*Optional AreaID.*
aCaptionPrefix	*Optional CaptionPrefix.*
aDeviceTypeID	*Optional DeviceTypeID.*
aModifiedDateTime	*Optional ModifiedDateTime*

Partial Match on CaptionPrefix. Match AFTER ModifiedDateTime.

Returns: List containing Devices.

List<DeviceType> **SMS.SMS_DB.GetDeviceTypes** ()

Retrieve Device Types.

Returns: List containing Device Types.

List<ManualOverride> **SMS.SMS_DB.GetManualOverrides** (
int? **aDeviceID** = null,
DateTime? **aModifiedDateTime** = null)

Retrieve Manual Overrides (MROs).

Parameters:

aDeviceID	*OptionalDeviceID*
aModifiedDateTime	*Optional ModifiedDateTime*

Returns: List containing MROs.

List<ManualOverrideSet> **SMS.SMS_DB.GetManualOverrideSets** ()

Retrieve ManualOverrideSets (MRO Sets).

Returns: List Containing MRO Sets.

List<TransactionCode> **SMS.SMS_DB.GetTransactionCodes** (
int? **aTransactionCodeID** = null,
int? **aTransactionCodeHi** = null)

Retrieve Transaction Codes.

Parameters:

- aTransactionCodeID *Optional TransactionCodeID*
- aTransactionCodeHi *Optional TransactionCodeHi*

Returns: List containing TransactionCodes.

List<TransactionGroup> **SMS.SMS_DB.GetTransactionGroups** (int? **aTransactionCodeHi** = null)

Retrieve Transaction Groups.

Parameters:

- aTransactionCodeHi *Optional TransactionCodeHi*

Returns: List containing Transaction Groups.

List<Transaction> **SMS.SMS_DB.GetTransactions** (
DateTime? **aStartTime** = null,
DateTime? **aEndTime** = null,
int? **aCardholderID** = null,
int? **aAreaID** = null,
int? **aDeviceID** = null)

Retrieve Logged Transactions.

Parameters:

aStartTime	*Optional Start Time.*
aEndTime	*Optional End Time.*
aCardholderID	*Optional CardholderID.*
aAreaID	*Optional AreaID.*
aDeviceID	*Optional DeviceID.*

Returns: List containing Transactions.

Hashtable **SMS.SMS_DB.GetUserDefinedFields** (int **aCardholderID)**

Retrieve All User Defined Field Values for Cardholder.
Column names are the hash keys. UDF values are hash values.

Parameters:

aCardholderID	*Return UDFs for CardholderID.*

Returns: Hash table Containing All User Defined Fields and values.

List<VideoCamera> **SMS.SMS_DB.GetVideoCameras** (
int? **VideoServerID** = null,
int? **DeviceID** = null,
int? **TranCodeHi** = null,
int? **TranCodeLo** = null,
int? **TranCodeID** = null)

Retrieve Configured Camera Control Entries.

Parameters:

VideoServerID	*Optional VideoServerID.*
DeviceID	*Optional DeviceID.*
TranCodeHi	*Optional TransactionCodeHi.*
TranCodeLo	*Optional TransactionCodeLo.*
TranCodeID	*Optional TransactionCodeID.*

Returns: List of Camera Control Items. *Only returns enabled items.*

List<VideoServer> **SMS.SMS_DB.GetVideoServers** ()

Retrieve Enabled Video Servers.

Returns: List containing enabled Video Servers.

bool **SMS.SMS_DB.InsertAlarmComment** (
int **aTrnHisID,**
string **aComment**,
string **aOperatorName**,
DateTime **aCommentDateTime**)

Insert an Alarm Comment.

Parameters:

aTrnHisID	*Transaction (Alarm) to Comment.*
aComment	*Comment Text.*
aOperatorNAme	*Operator Inserting Comment.*
aCommentDateTime	*Comment Timestamp (local to acknowledger).*

Returns: True if Insert Success

Property Documentation

- SqlConnection **SMS.SMS_DB.SQLConnection** [get]

  Connection object which may be used to run SQL queries directly.

SMS.ShuntStatusMessage Class Reference

Status Message explaining Report Shunting, Trigger Shunting or a Relay state.

Properties

- **ShuntStatusType status** `[get]`

  Shunt Status Type.

- **StatusReason reason** `[get]`

  Reason Item is shunted.

- **StatusDuration duration** `[get]`

  Shunt Duration. Not Recommended for Use.

Property Documentation

- StatusDuration **SMS.ShuntStatusMessage.duration** [get]

  Shunt Duration. Not Recommended for Use.

- StatusReason **SMS.ShuntStatusMessage.reason** [get]

  Reason Item is shunted.

- ShuntStatusType **SMS.ShuntStatusMessage.status** [get]

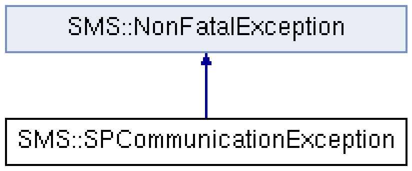
  Shunt Status Type.

SMS::SPCommunicationException Class Reference

Exception for signaling SP communication errors.

```
#include <SPCommWrapperBase.h>
```

Inheritance diagram for SMS::SPCommunicationException:

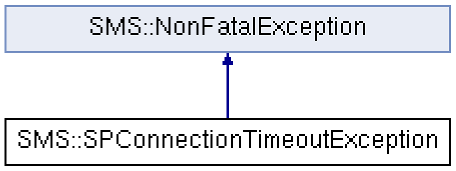


SMS::SPConnectionTimeoutException Class Reference

Exception for signaling that a timeout occurred waiting to connect to the SP.

```
#include <SPCommWrapperBase.h>
```

Inheritance diagram for SMS::SPConnectionTimeoutException:

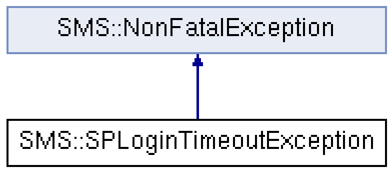


SMS::SPLoginTimeoutException Class Reference

Exception for signaling that a timeout occurred waiting to login to the SP.

```
#include <SPCommWrapperBase.h>
```

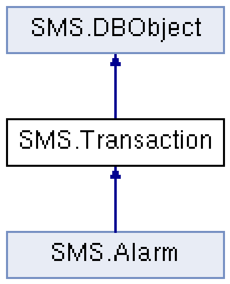
Inheritance diagram for SMS::SPLoginTimeoutException:



SMS.Transaction Class Reference

A **Transaction** is any event recorded by the system for future reporting and that may appear in the real time monitoring software.

Inheritance diagram for SMS.Transaction:



Public Attributes

- string **originalCardholderName**

  Cardholder Name at time of Transaction (if applicable). Typically Cardholder "First Name, Last Name".

Properties

- int **trnHisID** [get, set]

  Transaction Unique Identifier.
- TransactionCode **transactionCode** [get, set]

  TransactionCode for the Transaction.
- Area **area** [get, set]

  Area where Transaction occurred.
- Device **controller** [get, set]

  Parent Controller of the Device generating Transaction.
- Device **device** [get, set]

  Device generating the Transaction.
- Cardholder **cardholder** [get, set]

  Cardholder generating the Transaction (if applicable).
- int **issueCode** [get, set]

  Number physically encoded on Badge generating the Transaction (if applicable).
- int **cardReadData** [get, set]

  Reserved for Future Use.
- int **keyboardData** [get, set]

  Reserved for Future Use.

- int **floorNumber** [get, set]

  Reserved for Future Use.

- Device **workstation** [get, set]

  Workstation generating the Transaction (if applicable). Typically Operator login or MRO executed.

- string **operatorName** [get, set]

  Name of the Operator generating the Transaction (if applicable).
  Typically an Operator login or MRO executed.

- string **encodedID** [get, set]

  Encoded ID of the Badge generating the Transaction (if applicable).
  Typically access Transactions such as Access Granted or Access Denied.

- DateTime **transactionDateTime** [get, set]

  Local time the Transaction occurred.

Member Data Documentation

- string **SMS.Transaction.originalCardholderName**

  Cardholder Name at time of Transaction (if applicable). Typically Cardholder "First Name, Last Name".
  Useful when the cardholder record is deleted entirely, or record name was changed (perhaps due to marriage, or if the same Cardholder record was used for someone else).

Property Documentation

- Area **SMS.Transaction.area** [get, set]

  Area where Transaction occurred.

- Cardholder **SMS.Transaction.cardholder** [get, set]

  Cardholder generating the Transaction (if applicable).
  Typically only access Transactions. Unused in Transactions such as Contact activity, with no Cardholder associated.

- int **SMS.Transaction.cardReadData** [get, set]

  Reserved for Future Use.

- Device **SMS.Transaction.controller** [get, set]

  Parent Controller of the Device generating Transaction.
  Unused for Transactions like Operator logins, where there is no hardware Controller.

- Device **SMS.Transaction.device** [get, set]

  Device generating the Transaction.

- string **SMS.Transaction.encodedID** [get, set]

  Encoded ID of the Badge generating the Transaction (if applicable).
  Typically access Transactions such as Access Granted or Access Denied.
  A numeric value physically encoded on the Badge. Stored in the database as a string.

- int **SMS.Transaction.floorNumber** [get, set]

  Reserved for Future Use.

- int **SMS.Transaction.issueCode** [get, set]

  Number physically encoded on Badge generating the Transaction (if applicable).
  Typically only access Transactions. Set to 0 if the Badge had no Issue Code.

- int **SMS.Transaction.keyboardData** [get, set]

  Reserved for Future Use.

- string **SMS.Transaction.operatorName** [get, set]

  Name of the Operator generating the Transaction (if applicable).
  Typically an Operator login or MRO executed.

- TransactionCode **SMS.Transaction.transactionCode** [get, set]

  TransactionCode for the Transaction.

- DateTime **SMS.Transaction.transactionDateTime** [get, set]

  Local time the Transaction occurred.
  Always in local time; never GMT. Exists for All Transactions.

- int **SMS.Transaction.trnHisID** [get, set]

  Transaction Unique Identifier.

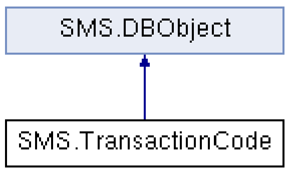
- Device **SMS.Transaction.workstation** [get, set]

  Workstation generating the Transaction (if applicable). Typically Operator login or MRO executed.

SMS.TransactionCode Class Reference

TransactionCode defines the type of transaction (i.e. a Valid Access **Transaction** versus a Contact Active transaction).

Inheritance diagram for SMS.TransactionCode:



Properties

- int **transactionCodeID** `[get, set]`

  TransactionCodeID is the unique identifier for a TransactionCode.
- int **transactionCodeHi** `[get, set]`

  TransactionCodeHi and TransactionCodeLo form a unique pair that indentify a TransactionCode.
- int **transactionCodeLo** `[get, set]`

  See TransactionCodeHi.
- string **caption** `[get, set]`

  TransactionCode description.

Property Documentation

- string **SMS.TransactionCode.caption** [get, set]

  TransactionCode description.
- int **SMS.TransactionCode.transactionCodeHi** [get, set]

  TransactionCodeHi and TransactionCodeLo form a unique pair that indentify a TransactionCode. Unlike TransactionCodeID, the TransactionCodeHi and TransactionCodeLo combination will always be identical at all SMS installations. TransactionCodeHi indicates which group of transactions this code belongs to, while TransactionCodeLo identifies the specific transaction.
- int **SMS.TransactionCode.transactionCodeID** [get, set]

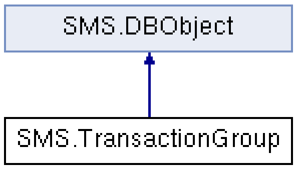
  TransactionCodeID is the unique identifier for a TransactionCode. TransactionCodeIDs may vary from one installation to another, (e.g. TransactionCodeID 144 might indicate "Access Denied - Area Access Privilege Not Yet Active" at one installation, but "Guest Added" at another).
- int **SMS.TransactionCode.transactionCodeLo** [get, set]

  See TransactionCodeHi.

SMS.TransactionGroup Class Reference

Transaction Groups indicate a class of transactions, such as a Valid Access transactions versus Contact Active transactions.

Inheritance diagram for SMS.TransactionGroup:



Properties

- int **transactionCodeHi** `[get, set]`

    TransactionGroup Unique Identifier.

- string **caption** `[get, set]`

    Transaction Group Description.

Property Documentation

- string **SMS.TransactionGroup.caption** [get, set]

    Transaction Group Description.

- int **SMS.TransactionGroup.transactionCodeHi** [get, set]

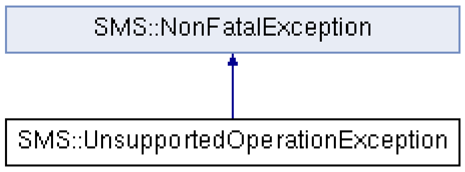
    TransactionGroup Unique Identifier.
    The naming convention is a legacy consideration, referring to the fact that one bit of the 32 highest order bits in a legacy structure is used by the controllers to identify the group of transactions for a TransactionCode. TransactionCodeHi identifier will be 1, 2, 4, 8, 16, etc. rather than a singly incrementing integer. Therefore, there can be no more than 32 Transaction Groups.

SMS::UnsupportedOperationException Class Reference

Exception for signaling that an operation is not supported (e.g. getting the status of a workstation **Device**.

```
#include <SPCommWrapperBase.h>
```

Inheritance diagram for SMS::UnsupportedOperationException:



SMS.VideoCamera Class Reference

Configuration Information for Video Cameras.

Properties

- int **controlID** `[get, set]`

  Camera Control Unique Identifier.
- string **controlCaption** `[get, set]`

  Camera Control Title.
- string **controlDescription** `[get, set]`

  Camera Control optional additional description.
- int **cameraNumber** `[get, set]`

  Camera Control Video Server Camera Identifier.
- int **transactionCodeHi** `[get, set]`

  Camera Control Transaction Group Unique Identifier.
- string **transactionGroup** `[get, set]`

  Transaction Group Description.
- int **selectedTransactionsSum** `[get, set]`

  Sum of Camera Control Selected Transactions.
- string **selectedTransactions** `[get, set]`

  Camera Control Selected Transactions Delimited String.
- int **timezoneID** `[get, set]`

  Unique TimezoneID of Camera Control.
- string **timezone** `[get, set]`

  Time zone Description.
- int **holidaySetID** `[get, set]`

  Unique HolidaySetID of Camera Control.
- string **holidaySet** `[get, set]`

  Holiday Set Description.
- int **deviceID** `[get, set]`

  Unique DeviceID of Camera Control.
- string **networkAddress** `[get, set]`

  Video Server IP Address or Host Name.
- int **videoServerModelID** `[get, set]`

  Unique VideoServerModelID for Camera Control.
- int **videoPreEvent** `[get, set]`

  Camera Control Time to Record Prior to Event (seconds).

- int **videoPostEvent** `[get, set]`

  Camera Control Time to Record After Event (seconds).
- int **cameraPosition** `[get, set]`

  Camera Control Preset Camera Position (if supported).

Property Documentation

- int **SMS.VideoCamera.cameraNumber** [get, set]

  Camera Control Video Server Camera Identifier.
- int **SMS.VideoCamera.cameraPosition** [get, set]

  Camera Control Preset Camera Position (if supported).
- string **SMS.VideoCamera.controlCaption** [get, set]

  Camera Control Title.
- string **SMS.VideoCamera.controlDescription** [get, set]

  Camera Control optional additional description.
- int **SMS.VideoCamera.controlID** [get, set]

  Camera Control Unique Identifier.
- int **SMS.VideoCamera.deviceID** [get, set]

  Unique DeviceID of Camera Control.
- string **SMS.VideoCamera.holidaySet** [get, set]

  Holiday Set Description.
- int **SMS.VideoCamera.holidaySetID** [get, set]

  Unique HolidaySetID of Camera Control.
- string **SMS.VideoCamera.networkAddress** [get, set]

  Video Server IP Address or Host Name.
- string **SMS.VideoCamera.selectedTransactions** [get, set]

  Camera Control Selected Transactions Delimited String.
- int **SMS.VideoCamera.selectedTransactionsSum** [get, set]

  Sum of Camera Control Selected Transactions.
- string **SMS.VideoCamera.timezone** [get, set]

  Time zone Description.
- int **SMS.VideoCamera.timezoneID** [get, set]

  Unique TimezoneID of Camera Control.
- int **SMS.VideoCamera.transactionCodeHi** [get, set]

  Camera Control Transaction Group Unique Identifier.
- string **SMS.VideoCamera.transactionGroup** [get, set]

  Transaction Group Description.

- int **SMS.VideoCamera.videoPostEvent** [get, set]

    Camera Control Time to Record After Event (seconds).

- int **SMS.VideoCamera.videoPreEvent** [get, set]

    Camera Control Time to Record Prior to Event (seconds).

- int **SMS.VideoCamera.videoServerModelID** [get, set]

    Unique VideoServerModelID for Camera Control.

SMS.VideoServer Class Reference

Configuration Information for SMS Video Servers.

Properties

- int **ModelID** `[get, set]`

  Video Server Model Unique Identifier.
- string **VideoServerModel** `[get, set]`

  Video Server Model Name.
- string **Description** `[get, set]`

  Video Server optional additional description.
- string **NetworkAddress** `[get, set]`

  IP Address or Host Name of Video Server.

Property Documentation

- string **SMS.VideoServer.Description** [get, set]

  Video Server optional additional description.
- int **SMS.VideoServer.ModelID** [get, set]

  Video Server Model Unique Identifier.
- string **SMS.VideoServer.NetworkAddress** [get, set]

  IP Address or Host Name of Video Server.
- string **SMS.VideoServer.VideoServerModel** [get, set]

  Video Server Model Name.

Packages

CHAPTER 4

Here are the packages with brief descriptions (if available):

- **SMS -** SMS Namespace includes all .NET elements of the SMS API
- **SPComm** - This namespace includes the lowest level wrappers around the SMS win32 SPComm.dll file

SMS

SMS Namespace includes all .NET elements of the SMS API.

Classes

- **DBObject -** *SMS_API objects with information stored in the SMS database inherit from this abstract class which provides the foundations for basic database I/O.*
- **TransactionCode -** *TransactionCode defines the type of transaction (i.e. a Valid Access Transaction versus a Contact Active transaction).*
- **TransactionGroup -** *Transaction Groups indicate a class of transactions, such as a Valid Access transactions versus Contact Active transactions.*
- **Area -** *An Area is an abstract container that may have zero, one or many devices assigned.*
- **AreaSet -** *An AreaSet is an abstract container that may have zero, one or many Areas assigned.*
- **AreaAccess -** *An AreaAccess record indicates access to a door.*
- **ManualOverride -** *A ManualOverride (MRO) is used to initiate a Device action (e.g. open a door, energize a relay, suspend contact reporting, etc).*
- **ManualOverrideSet -** *A ManualOverrideSet is a collection of MROs.*
- **DeviceType -** *Indicates the types of devices that can exist (Readers, Contacts, Workstations, etc).*
- **Device -** *Devices are things like Readers, Contacts, Workstations, etc.*
- **Credential -** *Credentials are items (typically badges) presented to a Reader to gain access.*
- **Cardholder -** *Cardholders are personnel interacting with card Reader Devices using security Badges.*
- **Transaction -** *A Transaction is any event recorded by the system for future reporting and that may appear in the real time monitoring software.*
- **AlarmCriteria -** *Information about the criteria that caused the Alarm.*
- **Operator -** *SMS application Operator.*
- **Alarm -** *Transaction designated as an Alarm; requires Operator attention.*
- **AlarmComment -** *Descriptive notes associated with an Alarm.*
- **AlarmInstruction -** *Instructions are associated to the criteria generating an Alarm. Alarms generated by the same criteria share Instructions.*
- **ShuntStatusMessage -** *Status Message explaining Report Shunting, Trigger Shunting or a Relay state.*
- **DeviceStatusMessage -** *Device Status Message.*
- **ContactStatusMessage -** *Contact Device Status Message.*
- **ReaderStatusMessage -** *Reader Device Status Message.*
- **RelayStatusMessage -** *Relay Device Status Message.*
- **VideoServer -** *Configuration Information for SMS Video Servers.*
- **VideoCamera -** *Configuration Information for Video Cameras.*
- **SMS_DB -** *This class abstracts database interaction and contains methods that return lists of DBObjects.*
- **SMS_API -** *Primary Class for SMS Access Control System Interaction. Documentation for many important methods are found in parent class SPCommWrapperBase.*
- **FatalException -** *Exception for signaling fatal errors.*
- **NonFatalException -** *Exception for signaling non fatal errors.*
- **SPCommunicationException -** *Exception for signaling SP communication errors.*
- **DBCommunicationException -** *Exception for signaling database communication errors.*

- **CIMCommunicationException -** *Exception for signaling CIM communication errors.*
- **DLLNotOpenException -** *Exception for signaling SchlagAPI.dll not open errors.*
- **DLLAlreadyOpenException -** *Exception for signaling SMS_API.dll already open errors.*
- **ControllerCommunicationException -** *Exception for signaling Controller communication errors.*
- **LibraryLoadingException -** *Exception for signaling library loading errors.*
- **LicenseCountExceededException -** *Exception for signaling that SMS_API is licensed, but the total number of workstation and/or API logins exceeds the number of licensed client logins.*
- **InvalidParameterException -** *Exception for signaling that one or more parameters passed to a routine are invalid.*
- **MemoryAllocationException -** *Exception for signaling that memory could not be allocated on the global heap.*
- **MemoryDeallocationException -** *Exception for signaling that memory could not be deallocated on the global heap.*
- **LicenseCountNotRetrievedException -** *Exception for signaling that an error occurred retrieving license count information from the SP.*
- **SPLoginTimeoutException -** *Exception for signaling that a timeout occurred waiting to login to the SP.*
- **DataNotLoadedException -** *Exception for signaling that internal reference data required by SMS_API was not successfully loaded.*
- **SPConnectionTimeoutException -** *Exception for signaling that a timeout occurred waiting to connect to the SP.*
- **LicenseInvalidException -** *Exception for signaling that SMS_API is not a licensed feature for this SMS installation.*
- **UnsupportedOperationException -** *Exception for signaling that an operation is not supported (e.g. getting the status of a workstation Device.*
- **abstract -** Clients should not instantiate this class. Use SMS::SMS_API.

Enumerations

- **StatusRequestType** {
  Reader,
  Relay,
  Contact }

  Status Request Type Enumeration.

- **ContactStatusType** {
  Closed,
  Open,
  SupervisedShorted,
  SupervisedOpen }

  Contact Status and Type Enumeration.

- **RelayStatusType** {
  Energized,
  Released }

  Relay Status Enumeration.

- **ShuntStatusType** {
  Normal,
  Shunted }

  Shunt Status Enumeration.

- **StatusReason** {
  Normal,
  EventTrigger,
  ScheduledOverride,
  ManualOverride }

  Status Reason Enumeration.

- **StatusDuration** {
  Timed,
  Indefinite }

  Status Duration Type Enumeration.

- **KeypadStatusType** {
  ReaderOnly,
  ReaderPlusKeypad }

  Keypad Status Type Enumeration.

Functions

- delegate void **TransactionHandler** (Transaction **aTransaction**)

  Implement this delegate to take action on any Transaction.

- delegate void **AlarmHandler** (Alarm **aAlarm**)

  Implement this delegate to take action on any Alarm.

- delegate void **AlarmAcknowledgementHandler** (
  int **aTrnHisID**,
  Operator **aAcknowledger**,
  DateTime **aAcknowledgedDateTime**)

  Implement this delegate to take action when an Alarm Acknowledged Notification is sent.

- delegate void **AlarmKillHandler** (int **aTrnHisID**)

  The System Processor sends this Notification to tell the SMS Alarm client application that an Alarm has been fully handled (secured and acknowledged, or forcefully killed) by the customer, and it should be removed from any real time displays.

- delegate void **DatabaseChangeHandler** (uint **aChangedDatabaseTablesBitmap**)

  Notification indicating that a database change has occurred.

- delegate void **DeviceStatusChangeHandler** (DeviceStatusMessage **aDeviceStatusMessage**)

  Device Status Change Notification.

- delegate void **MROExecutionCompleteHandler** (
  int **aTrnHisID**,
  int **aStatusCode**,
  String **aStatusMessage**)

  Invoked when an attempt to execute an MRO has completed.

- delegate void **ConnectionStatusChangedHandler** (
  bool **aConnectedToSP**,
  bool **aConnectedToDatabase**)

  Implement this callback to receive notifications whenever the status of the connection to the SP or database changes.

Enumeration Type Documentation

- enum **SMS::ContactStatusType**

  Contact Status and Type Enumeration. Indicates if the Contact is compromised (shorted or open) for supervised contacts.

Enumerator:

- **Closed** - Contact Closed. May or may not be the normal (or "secure") state, depending on whether the contact is normally open or closed.
- **Open** - Contact Open. May or may not be the normal (or "secure") state, depending on whether the contact is normally open or closed.
- **SupervisedShorted** - Circuit improperly shorted to ground. Contact supervision logic has detected interference.
- **SupervisedOpen** - Circuit improperly opened. Contact supervision logic has detected interference.

- **enum** SMS::KeypadStatusType

  Keypad Status Type Enumeration. Not simply an indicator of whether a keypad is present.
  The keypad must also be enabled.
  Some locations require a card and PIN during off peak hours, and require only a card during peak hours.
  In this case, the status would be ReaderOnly during peak hours, and ReaderPlusKeypad during off peak.
  Readers without a keypad associated will always return ReaderOnly.

Enumerator:

- **ReaderOnly** - Keypad Not Present OR Not Enabled.
- **ReaderPlusKeypad** - Keypad Present AND Enabled.

- enum **SMS::RelayStatusType**

  Relay Status Enumeration.

Enumerator:

- **Energized** - Relay Energized.
- **Released** - Relay Released.

- enum **SMS::ShuntStatusType**

  Shunt Status Enumeration.

Enumerator:

- **Normal** - Reporting or Trigger has not been shunted.
- **Shunted** - Reporting or Trigger is shunted.

- enum **SMS::StatusDuration**

  Status Duration Type Enumeration. Varies based on hardware support. Not Recommended for Use.

Enumerator:

- **Timed** - Status has a finite duration.
- **Indefinite** - Status has infinite duration.

- enum **SMS::StatusReason**

  Status Reason Enumeration.

Enumerator:

- **Normal** - Reporting or Trigger has not been shunted. Relay is in its normal state.
- **EventTrigger** - Reporting or Trigger is shunted, or the Relay is in its current state, due to an Event Trigger. A DOD shunted by a REX would generate an Event Trigger.
- **ScheduledOverride** - Reporting or Trigger is shunted, or the Relay is in its current state, due to a Schedule Event / Automatic Override. A hallway motion detector shunted during normal business hours would generate a Scheduled Override Trigger.
- **ManualOverride** - Reporting or Trigger is shunted, or the Relay is in its current state, due to an Operator action. An Operator executed MRO that performs a shunt would generate a Manual Override Trigger.

- enum **SMS::StatusRequestType**

  Status Request Type Enumeration.

Enumerator:

- **Reader -** Reader status.
- **Relay -** Relay status.
- **Contact -** Contact status.

Function Documentation

- delegate void **SMS.AlarmAcknowledgementHandler(**
  int **aTrnHisID**,
  Operator **aAcknowledger**,
  DateTime **aAcknowledgedDateTime**)

  Implement this delegate to take action when an Alarm Acknowledged Notification is sent.

- delegate void **SMS.AlarmHandler**(Alarm **aAlarm**)

  Implement this delegate to take action on any Alarm.
  Notification for the same Alarm may be received multiple times. If the Alarm is acknowledged or secured this handler will be invoked again. Clients should also handle it being called multiple times with the same Alarm data, even if the data has not changed.

- delegate void **SMS.AlarmKillHandler**(int **aTrnHisID**)

  The System Processor sends this Notification to tell the SMS Alarm client application that an Alarm has been fully handled (secured and acknowledged, or forcefully killed) by the customer, and it should be removed from any real time displays.

- delegate void **SMS.ConnectionStatusChangedHandler**(
  bool **aConnectedToSP**,
  bool **aConnectedToDatabase**)

  Implement this callback to receive notifications whenever the status of the connection to the SP or database changes.

  The very first time this callback is called is when the initial database connection either fails or succeeds. Clients may wish to detect this first call, and if the database is not connected, prompt their users to check that they have entered the server hostname information correctly, or check their network connectivity. Clients should be tolerant of receiving the same pair of statuses in succession, rather than assuming at least one of the bits will have changed for each notification.

- delegate void **SMS.DatabaseChangeHandler**(uint **aChangedDatabaseTablesBitmap**)

  Notification indicating that a database change has occurred.
  Clients caching data from the SMS DB should reload the cache.

- delegate void **SMS.DeviceStatusChangeHandler**(DeviceStatusMessage **aDeviceStatusMessage**)

  Device Status Change Notification.

- delegate void **SMS.MROExecutionCompleteHandler**(
  int **aTrnHisID**,
  int **aStatusCode**,
  String **aStatusMessage**)

  Invoked when an attempt to execute an MRO has completed.

Clients calling the ExecuteManualOverrideTask() method should expect this delegate to be invoked later to indicate the success or failure of the attempt. Notifications may NOT come in the same order that the MROs were issued. Status Code = 0 on success and the Status Message is empty.
A non-zero Status Code is returned on failure and the Status Message will contain information to help identify why the attempt failed. See the processor directives in the SPComm namespace for possible error codes. Failures typically contain either SP_DLL_CIM_ERROR or SP_DLL_CONTROLLER_ERROR.
Note the difference between an MRO Transaction and this notification.
Transactions are recorded in the history for future auditing and may be displayed in any connected SMS client. These notifications are only sent to the client who executed the MRO.

- delegate void **SMS.TransactionHandler**(Transaction **aTransaction**)

  Implement this delegate to take action on any Transaction.
  Typically, each Transaction is only sent once via this handler.

SPComm Namespace Reference

This namespace includes the lowest level wrappers around the **SMS** win32 SPComm.dll file.

Typedefs

- typedef unsigned int **SMS_BOOL**

  All Booleans are represented by a four byte unsigned integer.

Enumerations

- enum **SPMessageTypes** {
  SPMT_TRANSACTION = 0,
  SPMT_ALARM = 1,
  SPMT_KILL_ALARM = 5,
  SPMT_ACKNOWLEDGE_ALARM = 6,
  SPMT_DATABASE_CHANGE_NOTICE = 13,
  SPMT_DEVICE_STATUS = 19,
  SPMT_DEVICE_STATUS_REQUEST = 20,
  SPMT_VIEW_ALARM = 23 }

  The basic types of messages that may be received in the transaction callback function.

- enum **StatusRequestTypes** {
  SR_READER = 0,
  SR_RELAY = 1,
  SR_CONTACT = 2,
  SR_CONTROLLER = 3,
  SR_AREA = 4 }

  Indicates what type of device status message the client is receiving.

Device Status Request Type. Functions

- typedef void (__stdcall **_TSPTransactionCallbackFunction**)(
  unsigned int **aSPMessageType**,
  unsigned int **aTrnHisID**,
  unsigned int **aTransactionCodeID**,
  unsigned int **aAreaID**,
  unsigned int **aControllerID**,
  unsigned int **aDeviceID**,
  unsigned int **aCardholderID**,
  unsigned int **aIssueCode**,
  unsigned int **aCardReadData**,
  unsigned int **aKeyboardData**,
  unsigned int **aFloorNumber**,
  unsigned int **aWorkstationID**,
  unsigned int **aStatusRequestType**,
  unsigned int **aEncodedID**,
  unsigned int **aTransactionDateTime**,
  unsigned int **aDeviceStatus**,
  unsigned int **aSecured**,
  unsigned int **aSecuredDateTime**,
  unsigned int **aAcknowledged,**
  unsigned int **aAcknowledgedDateTime**,
  unsigned int **aAcknowledgerID**,
  unsigned int **aAlarmPriority**,
  unsigned int **aAlarmLabelID**,
  unsigned int **aChangedDatabaseTablesBitmap**)

  Callback that must be supplied by the client to receive Transaction events from the SMS_API.dll.

- int **OpenSPDll** (
  char* **aSPName**,
  TSPTransactionCallbackFunction **aSPTransactionCallbackFunction**,
  char* **aServername**,
  char* **aDatabaseName**,
  char* **aLoginName**,
  char* **aPassword**,
  TMROCallbackFunction **aMROCallbackFunction**,
  TConnectionStatusChangedCallbackFunction **aConnectionStatusChangedCallbackFunction**,
  int **aDatabaseReconnectIntervalInSeconds**,
  SMS_BOOL **aUseVerboseDebugging**,
  char* **aEventLogSource**)

  Opens SMS_API.dll and establishes connections to the System Processor (SP) and database.

- int **CloseSPDll** ()

  Closes SMS_API.dll and frees resources.

- int **SendTransaction** (
  int **aTrnHisID**,
  int **aTransactionCodeID**,
  SYSTEMTIME **aTransactionDateTime**,
  int **aSystemID**,
  int **aAreaID**,
  int **aControllerID**,
  int **aTransDeviceIDactionCodeID**,
  int **aCardholderID**,
  unsigned int **aEncodedID**,
  int **aIssueCode**,
  int **aCardreadData**,
  int **aKeyboardData**,
  int **aFloorNumber**,
  int **aWorkstationID**)

  Sends a transaction to the SP for distribution to clients.

- int **RequestStatus** (
  int **aDeviceID**,
  int **aStatusRequestType**,
  int **aControllerID**)

  Request Device status from the SP.

- int **SendDatabaseChange** (unsigned int **aDownloadTable**)

  Send a database change notice.

- int **ExecuteOverrideTask** (
  int **aOverrideTaskID**,
  int **aDuration**,
  char* **aOperatorName**)

  Send MRO Request to the SP.

- int **ExecuteOverrideSet** (
  int **aOverrideSetID**,
  int **aDuration**,
  char ***aOperatorName**)

  Send MRO Set Request to the SP.

- int **AcknowledgeAlarm** (
  int **aTrnHisID**,
  char* **aAcknowledgerName**)

  Acknowledge an Alarm.

- int **SendAntipassbackChange** (
  unsigned int **aEncodedID**,
  int **aState**)

  Change Badge Antipassback state.

Typedef Documentation

- typedef unsigned int **SPComm::SMS_BOOL**

  All Booleans are represented by a four byte unsigned integer.

Enumeration Type Documentation

- enum **SPComm::SPMessageTypes**

  The basic types of messages that may be received in the transaction callback function.

Enumerator:

- **SPMT_TRANSACTION -** A basic transaction, such as a valid access or relay energized.
- **SPMT_ALARM -** Contains information on a new Alarm transaction, or a previous Alarm transaction that has been updated (for instance, gone secure or been acknowledged).
- **SPMT_KILL_ALARM -** Clients should remove Alarms with the indicated TrnHisID from any monitor displays.
- **SPMT_ACKNOWLEDGE_ALARM -** The alarm indicated by TrnHisID has been acknowledged by another client. Only the TrnHisID, AcknowledgedDateTime and AcknowledgerID parameters are valid for these messages.
- **SPMT_DATABASE_CHANGE_NOTICE -** Data has changed in the database. Clients that cache data may wish to reload.
- **SPMT_DEVICE_STATUS -** Notification of a device's current status. These are sent specifically in response to a client's individual request for a device status. These are not sent continuously or automatically as a device status changes in real time.
- **SPMT_DEVICE_STATUS_REQUEST -** Clients should ignore any message of this type.
- **SPMT_VIEW_ALARM -** Clients should ignore any message of this type.

- enum **SPComm::StatusRequestTypes**

  Device Status Request Type.

Enumerator:

- **SR_READER -** Reader Status.
- **SR_RELAY -** Relay Status.
- **SR_CONTACT -** Contact Status.
- **SR_CONTROLLER -** Controller Status.
- **SR_AREA -** Area Status.

Function Documentation

- int **SPComm::AcknowledgeAlarm**(
  int **aTrnHisID**,
  char* **aAcknowledgerName**)

Acknowledge an Alarm.
This function will mark an Alarm as acknowledged, and record this information in the database.

Parameters:

aTrnHisID	*The TransactionID of the Alarm to acknowledge.*
aAcknowledgerName	*String to store as the person who acknowledged Alarm*

- int **SPComm::CloseSPDll**()

Closes SMS_API.dll and frees resources.
This function should be called when the client code is done communicating with the System Processor and database. It will terminate the network connection and free associated resources. See OpenSPDLL() for more information. The runtime MS Visual Studio debugger may incorrectly flag this as an error, giving a message of "Attempting managed execution inside OS Loader lock.". In this case, clients should disable the loader lock MDA while debugging.

- int **SPComm::ExecuteOverrideSet**(
  int **aOverrideSetID**,
  int **aDuration**,
  char* **aOperatorName**)

Send MRO Set Request to the SP.
This function will request that the SP execute a manual override set (a predefined group of manual overrides). Note that sometime subsequent to a client executing this function, they should expected to have their TMROCallbackFunction() invoked.

Parameters:

aOverrideSetID	*Unique Override Set ID to perform.*
aDuration	*MRO duration (if applicable). Most MROs have duration predefined, some may be timed "on the fly" by using this parameter. However, it is suggested that this parameter always be set to 0, since many types of controllers or door hardware door not support this feature.*
aOperatorName	*Operator name logged with MRO Set Transaction. If not supplied the default, "External System via API/DLL" will be used.*

Returns:	Unique Transaction History ID which identifies this attempt to execute an override. Used by the MRO callback function to identify which MRO execution succeeded or failed.

- int **SPComm::ExecuteOverrideTask**(
  int **aOverrideTaskID**,
  int **aDuration**,
  char* **aOperatorName**)

Send MRO Request to the SP.
This function will request that the SP execute a manual override. A manual override is an action defined for a device, such as momentarily opening a door or energizing an alarm siren relay for a period of time. Note that sometime subsequent to a client executing this function, they should expected to have their TMROCallbackFunction() invoked.

Parameters:

▪ aOverrideTaskID	*Unique Override Task ID to perform.*
▪ aDuration	*MRO duration (if applicable). Most MROs have duration predefined, some may be timed "on the fly" by using this parameter. However, it is suggested that this parameter always be set to 0, since many types of controllers or door hardware door not support this feature.*
▪ aOperatorName	*Operator name logged with MRO Set Transaction. If not supplied the default, "External System via API/DLL" will be used.*

Returns:	Unique Transaction History ID which identifies this attempt to execute an override. Used by the MRO callback function to identify which MRO has succeeded or failed.

- int **SPComm::OpenSPDll**(
  char* **aSPName**,
  TSPTransactionCallbackFunction **aSPTransactionCallbackFunction**,
  char* **aServername**,
  char* **aDatabaseName**,
  char* **aLoginName**,
  char* **aPassword**,
  TMROCallbackFunction **aMROCallbackFunction**,
  TConnectionStatusChangedCallbackFunction **aConnectionStatusChangedCallbackFunction**,
  int **aDatabaseReconnectIntervalInSeconds**,
  SMS_BOOL **aUseVerboseDebugging**,
  char* **aEventLogSource**)

Opens SMS_API.dll and establishes connections to the System Processor (SP) and database. This function is invoked by OpenSMS_API(). It will attempt to establish communication with the System Processor, as well as register a callback function defined by the client code used for event monitoring. If communication cannot be established initially with the System Processor, SMS_API will automatically and periodically try again; likewise, if communication is lost, it will attempt to handle this gracefully and reconnect when possible. This function should be used in tandem with the CloseSPDLL() function. Each Open invocation should eventually be terminated via a CloseSPDLL invocation. Typically, clients should call OpenSMS_API() during startup and CloseSPDLL() during shutdown. The behavior of SMS_API when this function is called successively multiple times is undefined. This function may be called multiple times within a single program execution as long as each call to Open is first followed by a CloseSPDLL(). One (unlikely, but plausible) reason to do this would be if the hostname of the SP changed during execution of the program.

Parameters:

▪ aSPName	*FQDN, IP Address or Hostname of the SP.*
▪ aSPTransactionCallbackFunction	*Function to be called when SMS_API receives event info from the SP.*
▪ aServername	*FQDN, IP Address or Hostname of the SMS SQL Server.*
▪ aDatabaseName	*Name of the SMS database.*
▪ aLoginName	*The SQL Server Login to use. Do not confuse with an SMS Operator Login ID.*
▪ aPassword	*The unencrypted password.*
▪ aMROCallbackFunction	*Function to be called when SMS_API receives information on MRO success or failure.*
▪ aConnectionStatusChangedCallbackFunction	*Function to be called when SMS_API receives SP or database connection status changes.*
▪ aDatabaseReconnectIntervalInSeconds	*Time between database reconnect attempts (0 = no reconnect attempts).*
▪ aUseVerboseDebugging	*Increase Logging. Additional information will be written to the the event log when true. Note that certain critical errors may be logged even when false.*

▪ aEventLogSource	*Errors and other information are written to the event log using this source name. Clients may supply a name. If null or an empty string is supplied, "SP_DLL" will be used.*

- int **SPComm::RequestStatus**(
  int **aDeviceID**,
  int **aStatusRequestType**,
  int **aControllerID**)

Request Device status from the SP.
This function will attempt to asynchronously determine the status of a Device.
The most common use of this function is to determine if a Contact point is active or secure (open / closed).
This function will send a request for status to the system; your status request callback function will later be invoked when the status is determined. Clients should implement a timeout in combination with this request, and interpret the Device as offline if they do not receive a response within the timeout period.

Parameters:

▪ aDeviceID	*Unique DeviceID of the Device whose status is requested.*
▪ aStatusRequestType	*Request type identifier. See StatusRequestTypes.*
▪ aControllerID	*The DeviceID of the controller to which the Device is attached.*

- int **SPComm::SendAntipassbackChange**(
  unsigned int **aEncodedID**,
  int **aState**)

Change Badge Antipassback state. Send Antipassback Change State to SP.

Parameters:

▪ aEncodedID	*EncodedID of the Badge to change antipassback state.*
▪ aState	*Desired antipassback state.*

- int **SPComm::SendDatabaseChange**(unsigned int **aDownloadTable**)

Send a database change notice.
This function sends a notice to the System Processor that the client has changed information in the database. The SP will then distribute this notice to other clients, allowing them to refresh their information from the database if necessary. A bitmapped value indicating the changed information must be provided. For instance an import job that only updates cardholder information would set this to be 0x1000. When in doubt all bits can be set, although this may cause client applications to needlessly reload their information from the database. The definitions are listed below, and they can be combined as needed using typical bitwise operators.

Parameters:

▪ aDownloadTable	*Bitmap indicating which information has been changed in the database.*

Bitmap Definitions:

- dld_ActionItem = $00000001

- dld_ActiveBadge = $00000002
- dld_AlarmAttachment = $00000004
- dld_Area = $00000008
- dld_AreaAccess = $00000010
- dld_AttachedCardholder = $00000020
- dld_AttachedDevice = $00000040
- dld_AvailableNumbers = $00000080
- dld_AvailableSitecodes = $00000100
- dld_Badge = $00000200
- dld_CallbackLink = $00000400
- dld_CallbackSet = $00000800
- dld_Cardholder = $00001000
- dld_Contact = $00002000
- dld_Controller = $00004000
- dld_Device = $00008000
- dld_EventTrigger = $00010000
- dld_Holiday = $00020000
- dld_HolidayLink = $00040000
- dld_HolidaySet = $00080000
- dld_OverrideAction = $00100000
- dld_OverrideTask = $00200000
- dld_Reader = $00400000
- dld_Relay = $00800000
- dld_RetiredBadge = $01000000
- dld_SitecodeLink = $02000000
- dld_SitecodeSet = $04000000
- dld_TimezoneInterval = $08000000
- dld_CameraControl = $10000000
- dld_DisplayControl = $20000000
- dld_AlarmGraphics = $40000000
- dld_GuestPass = $80000000
- dld_All = $FFFFFFFF

- int **SPComm::SendTransaction**(
  int **aTrnHisID**,
  int **aTransactionCodeID**,
  SYSTEMTIME **aTransactionDateTime**,
  int **aSystemID**,
  int **aAreaID**,
  int **aControllerID**,
  int **aTransDeviceIDactionCodeID**,
  int **aCardholderID**,
  unsigned int **aEncodedID**,
  int **aIssueCode**,
  int **aCardreadData**,
  int **aKeyboardData**,
  int **aFloorNumber**,
  int **aWorkstationID**)

Sends a transaction to the SP for distribution to clients.
This function will send a transaction to the SP (which will then send it along to any other transaction monitors, as well as apply alarm logic to it). It assumes the transaction has already been stored in the database using an appropriate stored procedure. See TSPTransactionCallbackFunction() for information on how to interpret parameters. Note that this only distributes the transaction to the real time messaging system. It does not record the transaction to the database.

- typedef **SPComm::void**(__stdcall **_TConnectionStatusChangedCallbackFunction**)(
  SMS_BOOL **aConnectedToSP**,
  SMS_BOOL **aConnectedToDatabase**,
  SMS_BOOL **aEncounteredError**,
  char* **aLastErrorMessage**)

Callback that must be supplied by the client to receive connection status changed notifications from SMS_API.dll.

Parameters:

▪ aConnectedToSP	*SP connection status.*
▪ aConnectedToDatabase	*Database connection status.*
▪ aEncounteredError	*Fatal error occurred.* *True if the SMS_API.dll encountered a fatal error during load.* *Recommended to terminate the client process in this case.* *Alternately, CloseSPDLL and OpenSPDLL may be attempted;* *but continued use of SMS_API in this state is not recommended.*
▪ aLastErrorMessage	*Additional error details, if available when aEncounteredError is true.* *Clients should not keep a shallow (reference) copy of this string,* *as it may be freed independently. If it needs to be stored,* *clients should make a deep copy of this string*

File

CHAPTER 5

File List

Here is a list of all documented files with brief descriptions:

- **HelpMainPage.h**
- **SMS_API.cs -**This is the main file for the SMS .NET Communications API
- **SPComm.h** This is the header file used to wrap the SMS Win32 DLL from C++ in visual studio
- **SPCommWrapperBase.h -**This is a C++ .NET (CLI) wrapper around the SPComm.h file

SMS_API.cs File Reference

This is the main file for the **SMS** .NET SMS Communications API.

Classes

- **SMS.DBObject -** *SMS_API objects with information stored in the SMS database inherit from this abstract class which provides the foundations for basic database I/O.*
- **SMS.TransactionCode -** *TransactionCode defines the type of transaction (i.e. a Valid Access Transaction versus a Contact Active transaction).*
- **SMS.TransactionGroup -** *Transaction Groups indicate a class of transactions, such as a Valid Access transactions versus Contact Active transactions.*
- **SMS.Area -** *An Area is an abstract container that may have zero, one or many devices assigned.*
- **SMS.AreaSet -** *An AreaSet is an abstract container that may have zero, one or many Areas assigned.*
- **SMS.AreaAccess -** *An AreaAccess record indicates access to a door.*
- **SMS.ManualOverride -** *A ManualOverride (MRO) is used to initiate a Device action (e.g. open a door, energize a relay, suspend contact reporting, etc).*
- **SMS.ManualOverrideSet -** *A ManualOverrideSet is a collection of MROs.*
- **SMS.DeviceType -** *Indicates the types of devices that can exist (Readers, Contacts, Workstations, etc).*
- **SMS.Device -** *Devices are things like Readers, Contacts, Workstations, etc.*
- **SMS.Credential -** *Credentials are items (typically badges) presented to a Reader to gain access.*
- **SMS.Cardholder -** *Cardholders are personnel interacting with card Reader Devices using security Badges.*
- **SMS.Transaction -** *A Transaction is any event recorded by the system for future reporting and that may appear in the real time monitoring software.*
- **SMS.AlarmCriteria -** *Information about the criteria that caused the Alarm.*
- **SMS.Operator -** *SMS application Operator.*
- **SMS.Alarm -** *Transaction designated as an Alarm; requires Operator attention.*
- **SMS.AlarmComment -** *Descriptive notes associated with an Alarm.*
- **SMS.AlarmInstruction -** *Instructions are associated to the criteria generating an Alarm. Alarms generated by the same criteria share Instructions.*
- **SMS.ShuntStatusMessage -** *Status Message explaining Report Shunting, Trigger Shunting or a Relay state.*
- **SMS.DeviceStatusMessage -** *Device Status Message.*
- **SMS.ContactStatusMessage -** *Contact Device Status Message.*
- **SMS.ReaderStatusMessage -** *Reader Device Status Message.*
- **SMS.RelayStatusMessage -** *Relay Device Status Message.*
- **SMS.VideoServer -** *Configuration Information for SMS Video Servers.*
- **SMS.VideoCamera -** *Configuration Information for Video Cameras.*
- **SMS.SMS_DB -** *This class abstracts database interaction and contains methods that return lists of DBObjects.*
- **SMS.SMS_API -** *Primary Class for SMS Access Control System Interaction. Documentation for many important methods are found in parent class SPCommWrapperBase.*

Packages

- **SMS -** *SMS Namespace includes all .NET elements of the SMS API.*

Enumerations

- **SMS.StatusRequestType** {
  SMS.Reader,
  SMS.Relay,
  SMS.Contact }

  Status Request Type Enumeration.

- **SMS.ContactStatusType** {
  SMS.Closed,
  SMS.Open,
  SMS.SupervisedShorted,
  SMS.SupervisedOpen }

  Contact Status and Type Enumeration.

- **SMS.RelayStatusType** {
  SMS.Energized,
  SMS.Released }

  Relay Status Enumeration.

- **SMS.ShuntStatusType** {
  SMS.Normal,
  SMS.Shunted }

  Shunt Status Enumeration.

- **SMS.StatusReason** {
  SMS.Normal,
  SMS.EventTrigger,
  SMS.ScheduledOverride,
  SMS.ManualOverride }

  Status Reason Enumeration.

- **SMS.StatusDuration** {
  SMS.Timed,
  SMS.Indefinite }

  Status Duration Type Enumeration.

- **SMS.KeypadStatusType** {
  SMS.ReaderOnly,
  SMS.ReaderPlusKeypad }

  Keypad Status Type Enumeration.

Functions

- delegate void **SMS.TransactionHandler** (Transaction **aTransaction**)

  Implement this delegate to take action on any Transaction.

- delegate void **SMS.AlarmHandler** (Alarm **aAlarm**)

  Implement this delegate to take action on any Alarm.

- delegate void **SMS.AlarmAcknowledgementHandler** (
  int **aTrnHisID**,
  Operator **aAcknowledger**,
  DateTime **aAcknowledgedDateTime**)

  Implement this delegate to take action when an Alarm Acknowledged Notification is sent.

- delegate void **SMS.AlarmKillHandler** (int **aTrnHisID**)

  The System Processor sends this Notification to tell the SMS Alarm client application that an Alarm has been fully handled (secured and acknowledged, or forcefully killed) by the customer, and it should be removed from any real time displays.

- delegate void **SMS.DatabaseChangeHandler** (uint **aChangedDatabaseTablesBitmap**)

  Notification indicating that a database change has occurred.

- delegate void **SMS.DeviceStatusChangeHandler** (DeviceStatusMessage **aDeviceStatusMessage**)

  Device Status Change Notification.

- delegate void **SMS.MROExecutionCompleteHandler** (
  int **aTrnHisID**,
  int **aStatusCode**,
  String **aStatusMessage**)

  Invoked when an attempt to execute an MRO has completed.

- delegate void **SMS.ConnectionStatusChangedHandler** (
  bool **aConnectedToSP**,
  bool **aConnectedToDatabase**)

  Implement this callback to receive notifications whenever the status of the connection to the SP or database changes.

SPCommWrapperBase.h File Reference

This is a C++ .NET (CLI) wrapper around the **SPComm.h** file.

```
#include "SPComm.h"
```

Classes

- **SMS::FatalException -** *Exception for signaling fatal errors.*
- **SMS::NonFatalException -** *Exception for signaling non fatal errors.*
- **SMS::SPCommunicationException -** *Exception for signaling SP communication errors.*
- **SMS::DBCommunicationException -** *Exception for signaling database communication errors.*
- **SMS::CIMCommunicationException -** *Exception for signaling CIM communication errors.*
- **SMS::DLLNotOpenException -** *Exception for signaling SchlagAPI.dll not open errors.*
- **SMS::DLLAlreadyOpenException -** *Exception for signaling SMS_API.dll already open errors.*
- **SMS::ControllerCommunicationException -** *Exception for signaling Controller communication errors.*
- **SMS::LibraryLoadingException -** *Exception for signaling library loading errors.*
- **SMS::LicenseCountExceededException -** *Exception for signaling that SMS_API is licensed, but the total number of workstation and/or API logins exceeds the number of licensed client logins.*
- **SMS::InvalidParameterException -** *Exception for signaling that one or more parameters passed to a routine are invalid.*
- **SMS::MemoryAllocationException -** *Exception for signaling that memory could not be allocated on the global heap.*
- **SMS::MemoryDeallocationException -** *Exception for signaling that memory could not be deallocated on the global heap.*
- **SMS::LicenseCountNotRetrievedException -** *Exception for signaling that an error occurred retrieving license count information from the SP.*
- **SMS::SPLoginTimeoutException -** *Exception for signaling that a timeout occurred waiting to login to the SP.*
- **SMS::DataNotLoadedException -** *Exception for signaling that internal reference data required by SMS_API was not successfully loaded.*
- **SMS::SPConnectionTimeoutException -** *Exception for signaling that a timeout occurred waiting to connect to the SP.*
- **SMS::LicenseInvalidException -** *Exception for signaling that SMS_API is not a licensed feature for this SMS installation.*
- **SMS::UnsupportedOperationException -** *Exception for signaling that an operation is not supported (e.g. getting the status of a workstation Device.*
- **SMS::abstract -** *Clients should not instantiate this class. Use SMS::SMS_API.*

Packages

- **SMS** - *SMS Namespace includes all .NET elements of the SMS API.*

SPComm.h File Reference

This is the header file used to wrap the **SMS** Win32 DLL from C++ in visual studio.

```
#include "stdafx.h"

#include <Windows.h>
```

Packages

- namespace **SPComm** **-** *This namespace includes the lowest level wrappers around the SMS win32 SPComm.dll file.*

Defines

- #define **SP_DLL_SUCCESS** 0

  Returned by functions when operation completes normally.

- #define **SP_DLL_UNKNOWN_ERROR** -1

  Returned when an unknown error is encountered.
  Clients should not use the library after receiving this return code.

- #define **SP_DLL_SP_ERROR** -2

  Returned when clients try to invoke a function while the SP is disconnected, or upon some other non fatal error encountered while communicating with the SP.

- #define **SP_DLL_DB_ERROR** -3

  Returned when clients try to invoke a function while the DB is disconnected, or upon some other non fatal error encountered while querying the database.

- #define **SP_DLL_CIM_ERROR** -4

  Returned when clients try to invoke a CIM related function (such as executing an MRO) and the SMS API DLL encounters a non fatal error while attempting to communicate to a CIM.

- #define **SP_DLL_NOT_OPEN_ERROR** -5

  Returned when clients try to invoke a function before calling OpenSPDLL or after calling CloseSPDLL.

- #define **SP_DLL_ALREADY_OPEN_ERROR** -6

  Returned when clients call OpenSPDLL and it is already open.

- #define **SP_DLL_CONTROLLER_ERROR** -7

  Returned when there is some error communicating to a controller. Typically this would only be seen in an MRO callback function.

- #define **SP_DLL_LIBRARY_LOADING_ERROR** -8

  Returned when there is some error within the DLL wrapper code. Usually indicates the DLL is missing, wrong version, misnamed, etc.

- #define **API_RETCODE_ERROR_LICENSE_COUNT_EXCEEDED** -9

  Returned when SMS_API is licensed, but number of total number of client logins (including SMS_API) exceeds the number of licensed client logins.

- #define **API_RETCODE_ERROR_INVALID_PARAMETER** -10

  Returned when one or more parameters passed to a routine are invalid.

- #define **API_RETCODE_ERROR_MEMORY_ALLOC** -11

  Returned when memory could not be allocated on the global heap.

- #define **API_RETCODE_ERROR_MEMORY_DEALLOC** -12

  Returned when memory could not be deallocated (released) to the global heap.

- #define **API_RETCODE_ERROR_LICENSE_COUNT_NOT_RETRIEVED** -13

  Returned when an error occurred retrieving license count information from the System Processor.

- #define **API_RETCODE_ERROR_TIMEOUT_LOGIN_SP** -14

  Returned when a timeout occurred waiting to login to the System Processor.

- #define **API_RETCODE_ERROR_DATA_NOT_SUCCESSFULLY_LOADED** -15

  Returned when internal reference data required by SMS_API was not successfully loaded.

- #define **API_RETCODE_ERROR_TIMEOUT_CONNECTION_SP** -16

  Returned when timeout occurs waiting to connect to the System Processor.

- #define **API_RETCODE_ERROR_LICENSE_INVALID** -17

  Returned when the SMS_API is not a licensed feature for the current SMS installation.

Typedefs

- typedef unsigned int **SPComm::SMS_BOOL**

  All Booleans are represented by a four byte unsigned integer.

Enumerations

- **SPComm::SPMessageTypes** {
  SPComm::SPMT_TRANSACTION = 0,
  SPComm::SPMT_ALARM = 1,
  SPComm::SPMT_KILL_ALARM = 5,
  SPComm::SPMT_ACKNOWLEDGE_ALARM = 6,
  SPComm::SPMT_DATABASE_CHANGE_NOTICE = 13,
  SPComm::SPMT_DEVICE_STATUS = 19,
  SPComm::SPMT_DEVICE_STATUS_REQUEST = 20,
  SPComm::SPMT_VIEW_ALARM = 23 }

  The basic types of messages that may be received in the transaction callback function.

- **SPComm::StatusRequestTypes** {
  SPComm::SR_READER = 0,
  SPComm::SR_RELAY = 1,
  SPComm::SR_CONTACT = 2,
  SPComm::SR_CONTROLLER = 3,
  SPComm::SR_AREA = 4 }

  Device Status Request Type.

Functions

- typedef **SPComm::void** (__stdcall **_TSPTransactionCallbackFunction**)(
  unsigned int **aSPMessageType**,
  unsigned int **aTrnHisID**,
  unsigned int **aTransactionCodeID**,
  unsigned int **aAreaID**,
  unsigned int **aControllerID**,
  unsigned int **aDeviceID**,
  unsigned int **aCardholderID**,
  unsigned int **aIssueCode**,
  unsigned int **aCardReadData**,
  unsigned int **aKeyboardData**,
  unsigned int **aFloorNumber**,
  unsigned int **aWorkstationID**,
  unsigned int **aStatusRequestType**,
  unsigned int **aEncodedID**,
  unsigned int **aTransactionDataTime**,
  unsigned int **aDeviceStatus**,
  unsigned int **aSecured**,
  unsigned int **aSecuredDateTime**,
  unsigned int **aAcknowledged**,
  unsigned int **aAcknowledgedDateTime**,
  unsigned int **aAcknowledgerID**,
  unsigned int **aAlarmPriority**,
  unsigned int **aAlarmLabelID**,
  unsigned int **aChangedDatabaseTablesBitmap**)

  Callback that must be supplied by the client to receive Transaction events from the SMS_API.dll.

- int **SPComm::OpenSPDll** (
  char* **aSPName**,
  TSPTransactionCallbackFunction **aSPTransactionCallbackFunction**,
  char* **aServername**,
  char* **aDatabaseName**,
  char* **aLoginName**,
  char* **aPassword**,
  TMROCallbackFunction **aMROCallbackFunction**,
  TConnectionStatusChangedCallbackFunction **aConnectionStatusChangedCallbackFunction**,
  int **aDatabaseReconnectIntervalInSeconds**,
  SMS_BOOL **aUseVerboseDebugging**,
  char* **aEventLogSource**)

  Opens SMS_API.dll and establishes connections to the System Processor (SP) and database.

- int **SPComm::CloseSPDll** ()

  Closes SMS_API.dll and frees resources.

- int **SPComm::SendTransaction** (
  int **aTrnHisID**,
  int **aTransactionCodeID**,
  SYSTEMTIME **aTransactionDateTime**,
  int **aSystemID**,
  int **aAreaID**,
  int **aControllerID**,
  int **aTransDeviceIDactionCodeID**,
  int **aCardholderID**,
  unsigned int **aEncodedID**,
  int **aIssueCode**,
  int **aCardreadData**,
  int **aKeyboardData**,
  int **aFloorNumber**,
  int **aWorkstationID**)

  Sends a transaction to the SP for distribution to clients.

- int **SPComm::RequestStatus** (
  int **aDeviceID**,
  int **aStatusRequestType**,
  int **aControllerID**)

  Request Device status from the SP.

- int **SPComm::SendDatabaseChange** (unsigned int **aDownloadTable**)

  Send a database change notice.

- int **SPComm::ExecuteOverrideTask** (
  int **aOverrideTaskID**,
  int aDuration,
  char* **aOperatorName**)

  Send MRO Request to the SP.

- int **SPComm::ExecuteOverrideSet** (
  int **aOverrideSetID**,
  int **aDuration**,
  char* **aOperatorName**)

  Send MRO Set Request to the SP.

- int **SPComm::AcknowledgeAlarm** (
  int **aTrnHisID**,
  char* **aAcknowledgerName**)

  Acknowledge an Alarm.

- int **SPComm::SendAntipassbackChange** (
  unsigned int **aEncodedID**,
  int **aState**)

  Change Badge Antipassback state.

Define Documentation

- #define **API_RETCODE_ERROR_DATA_NOT_SUCCESSFULLY_LOADED** -15

  Returned when internal reference data required by SMS_API was not successfully loaded.

- #define **API_RETCODE_ERROR_INVALID_PARAMETER** -10

  Returned when one or more parameters passed to a routine are invalid.

- #define **API_RETCODE_ERROR_LICENSE_COUNT_EXCEEDED** -9

  Returned when SMS_API is licensed, but number of total number of client logins (including SMS_API) exceeds the number of licensed client logins.

- #define **API_RETCODE_ERROR_LICENSE_COUNT_NOT_RETRIEVED** -13

  Returned when an error occurred retrieving license count information from the System Processor.

- #define **API_RETCODE_ERROR_LICENSE_INVALID** -17

  Returned when the SMS_API is not a licensed feature for the current SMS installation.

- #define **API_RETCODE_ERROR_MEMORY_ALLOC** -11

  Returned when memory could not be allocated on the global heap.

- #define **API_RETCODE_ERROR_MEMORY_DEALLOC** -12

  Returned when memory could not be deallocated (released) to the global heap.

- #define **API_RETCODE_ERROR_TIMEOUT_CONNECTION_SP** -16

  Returned when timeout occurs waiting to connect to the System Processor.

- #define **API_RETCODE_ERROR_TIMEOUT_LOGIN_SP** -14

  Returned when a timeout occurred waiting to login to the System Processor.

- #define **SP_DLL_ALREADY_OPEN_ERROR** -6

  Returned when clients call OpenSPDLL and it is already open.

- #define **SP_DLL_CIM_ERROR** -4

  Returned when clients try to invoke a CIM related function (such as executing an MRO) and the SMS API DLL encounters a non fatal error while attempting to communicate to a CIM.

- #define **SP_DLL_CONTROLLER_ERROR** -7

  Returned when there is some error communicating to a controller.
  Typically this would only be seen in an MRO callback function.

- #define **SP_DLL_DB_ERROR** -3

  Returned when clients try to invoke a function while the DB is disconnected, or upon some other non fatal error encountered while querying the database.

- #define **SP_DLL_LIBRARY_LOADING_ERROR** -8

  Returned when there is some error within the DLL wrapper code. Usually indicates the DLL is missing, wrong version, misnamed, etc.

- #define **SP_DLL_NOT_OPEN_ERROR** -5

  Returned when clients try to invoke a function before calling OpenSPDLL or after calling CloseSPDLL.

- #define **SP_DLL_SP_ERROR** -2

  Returned when clients try to invoke a function while the SP is disconnected, or upon some other non fatal error encountered while communicating with the SP.

- #define **SP_DLL_SUCCESS** 0

  Returned by functions when operation completes normally.

- #define **SP_DLL_UNKNOWN_ERROR** -1

  Returned when an unknown error is encountered.
  Clients should not use the library after receiving this return code.
  Clients may attempt to call CloseSPDLL and then OpenSPDLL, but it is instead recommended that the process exit.

Implementation Demo Application

CHAPTER 6

Introduction

The Implementation Demo Application allows the user to test the API functions. Once a connection is established it will show all transactions in the system as well as allow the user to read and execute various functions of the API and SMS.

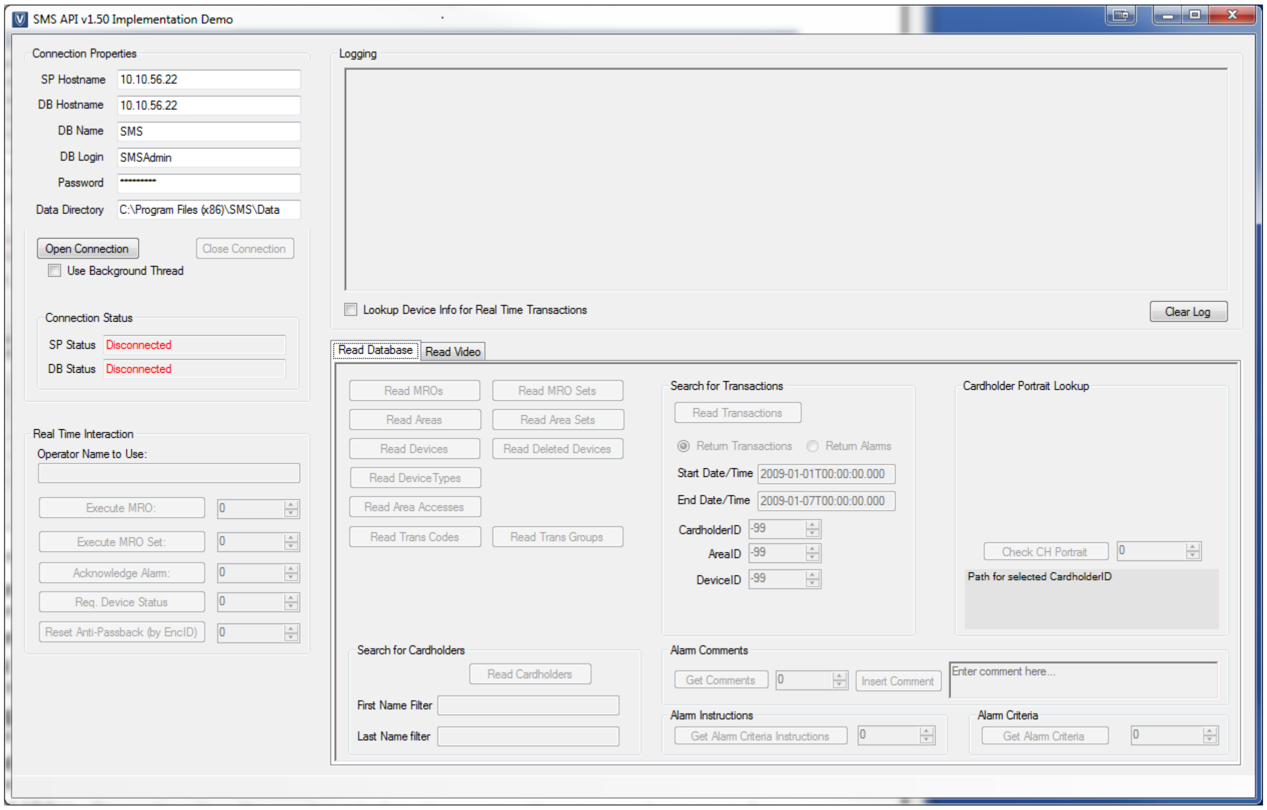
Accessing the Application

To start the Implementation Demo Application:

1. Go to the **Help** folder included with the installation of the API.
2. Open the **SampleApps** folder.
3. Open the **SMS_API_Demonstration** folder.
4. Open the **bin** folder.
5. Open the **Release** folder.
6. Run the **SMS_API_Demo.exe** file. The Implementation Demo Application will open.

Overview

When the Implementation Demo application is first opened the following screen will be displayed.


SMS API v1.50 Implementation Demo

Connection Properties

SP Hostname: 10.10.56.22
DB Hostname: 10.10.56.22
DB Name: SMS
DB Login: SMSAdmin
Password: *********
Data Directory: C:\Program Files (x86)\SMS\Data

Use Background Thread

Connection Status

SP Status: Disconnected
DB Status: Disconnected

Real Time Interaction

Operator Name to Use:

Logging

Lookup Device Info for Real Time Transactions

Search for Transactions

Return Transactions / Return Alarms
Start Date/Time: 2009-01-01T00:00:00.000
End Date/Time: 2009-01-07T00:00:00.000
CardholderID: -99
AreaID: -99
DeviceID: -99

Cardholder Portrait Lookup

Path for selected CardholderID

Search for Cardholders

First Name Filter
Last Name filter

Alarm Comments

Enter comment here...

Alarm Instructions

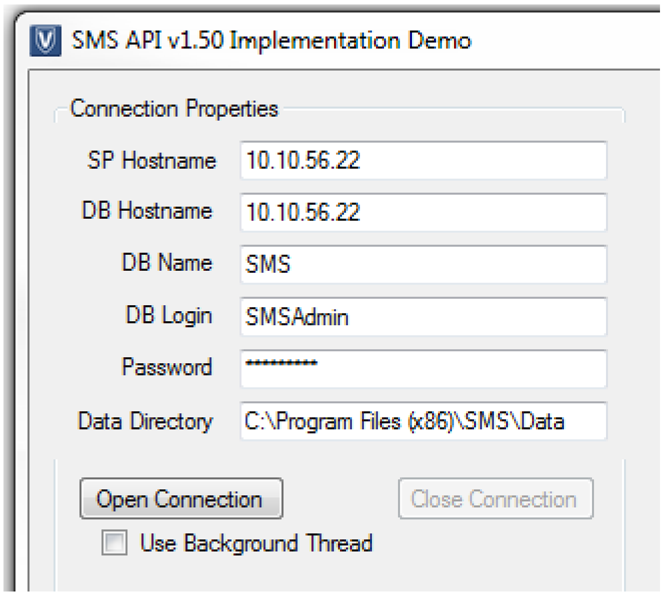
Alarm Criteria

This is the main window of the application and is broken up into the following areas:

- **Connection Properties** - This is where the SMS server information is input and connections to the SMS system are established.
- **Connection Status** - This section displayes the connection status and error messages.
- **Real Time Interaction** - This section allows the user to execute various features of SMS in order to test the connection.
- **Logging** - This is the display window of the application. It will show all the various types of information generated by SMS and the application.
- **Read Database** - This section allows the user to read various features of SMS in order to test the connection.
- **Read Video** - This section allows the user to read information on any video servers and cameras in the system.

Connection Properties

The connection properties section is where the user enters all the SMS information.


SMS API v1.50 Implementation Demo

Connection Properties

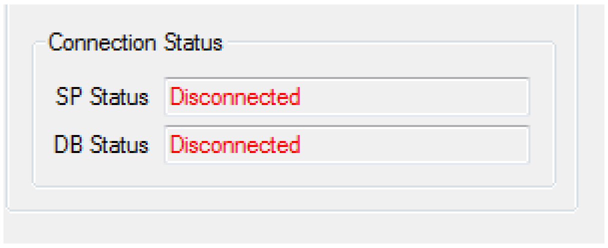
SP Hostname	10.10.56.22
DB Hostname	10.10.56.22
DB Name	SMS
DB Login	SMSAdmin
Password	**********
Data Directory	C:\Program Files (x86)\SMS\Data

Enter all requested information.

- **Open Connection** - Click to establish a connection with SMS.
- **Close Connection** - Click to cease communication with SMS.
- **Use Background Thread** - The API will be instantiated on a background thread instead of the Implementation Demo main thread.

Connection Status

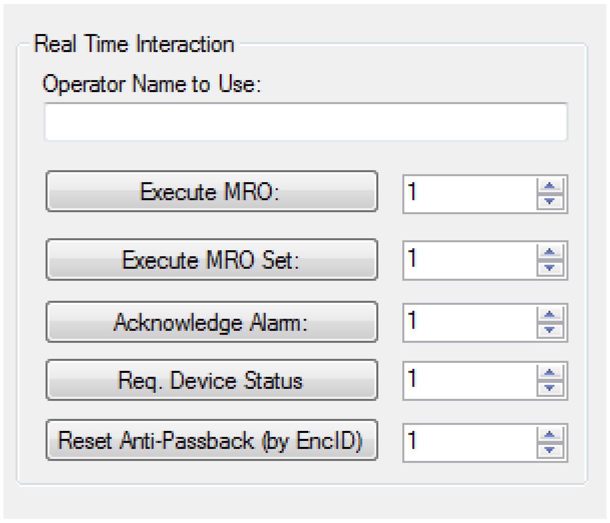
This section displays the connection status with the System Processor (SP) and the Database (DB). Connections to **both** the SP and DB must succeed for API usage.


Connection Status

SP Status	Disconnected
DB Status	Disconnected

Real Time Interaction

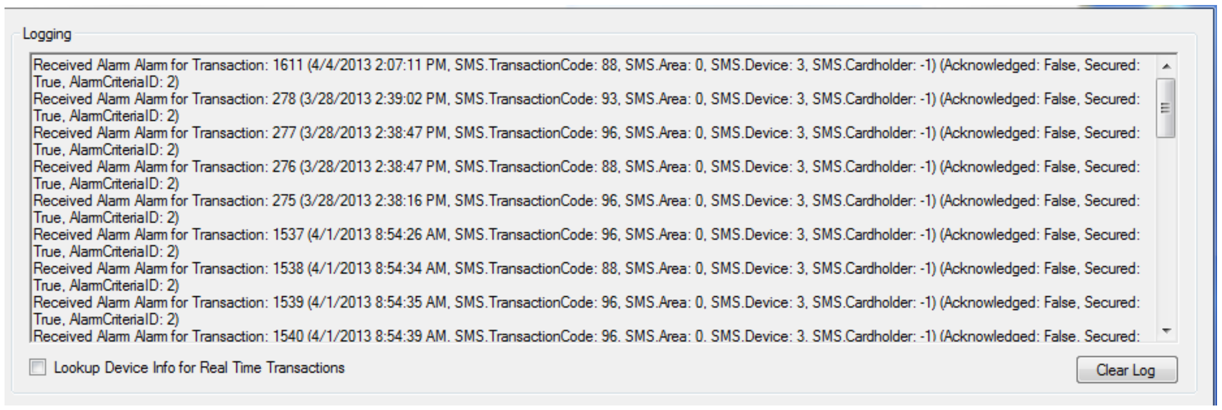
The Real Time Interaction section allows the user to execute various functions of SMS.



- **Operator Name to Use** - Enter the Operator name.
- **Execute functions** - Click on the desired button to execute the listed function. Buttons initially disabled until appropriate ID selected.
- **Drop Down Option** - Use the drop down boxes to specify the ID number of the desired function (MRO ID, MRO Set ID, Alarm ID, etc.)

Logging

The logging window is used to display the various information in the system. By default it automatically shows any transactions in the system in real time. It will also display the specific information of various functions (Cardholders, Alarms, MROs, etc.) when an execute or read command is sent by the user.

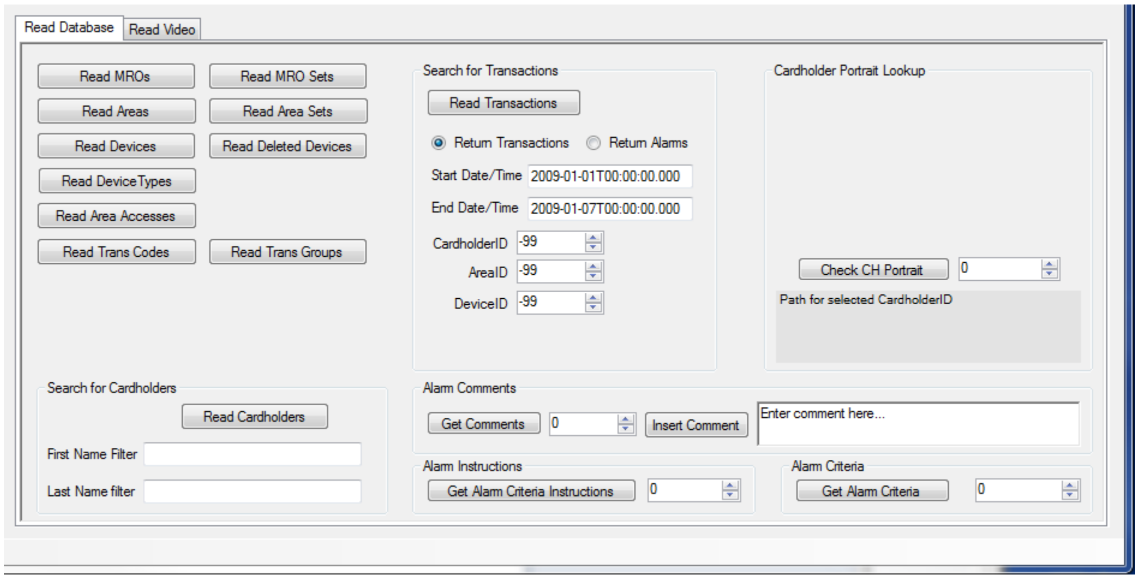


```
Received Alarm Alarm for Transaction: 1611 (4/4/2013 2:07:11 PM, SMS.TransactionCode: 88, SMS.Area: 0, SMS.Device: 3, SMS.Cardholder: -1) (Acknowledged: False, Secured:
True, AlarmCriteriaID: 2)
Received Alarm Alarm for Transaction: 278 (3/28/2013 2:39:02 PM, SMS.TransactionCode: 93, SMS.Area: 0, SMS.Device: 3, SMS.Cardholder: -1) (Acknowledged: False, Secured:
True, AlarmCriteriaID: 2)
Received Alarm Alarm for Transaction: 277 (3/28/2013 2:38:47 PM, SMS.TransactionCode: 96, SMS.Area: 0, SMS.Device: 3, SMS.Cardholder: -1) (Acknowledged: False, Secured:
True, AlarmCriteriaID: 2)
Received Alarm Alarm for Transaction: 276 (3/28/2013 2:38:47 PM, SMS.TransactionCode: 88, SMS.Area: 0, SMS.Device: 3, SMS.Cardholder: -1) (Acknowledged: False, Secured:
True, AlarmCriteriaID: 2)
Received Alarm Alarm for Transaction: 275 (3/28/2013 2:38:16 PM, SMS.TransactionCode: 96, SMS.Area: 0, SMS.Device: 3, SMS.Cardholder: -1) (Acknowledged: False, Secured:
True, AlarmCriteriaID: 2)
Received Alarm Alarm for Transaction: 1537 (4/1/2013 8:54:26 AM, SMS.TransactionCode: 96, SMS.Area: 0, SMS.Device: 3, SMS.Cardholder: -1) (Acknowledged: False, Secured:
True, AlarmCriteriaID: 2)
Received Alarm Alarm for Transaction: 1538 (4/1/2013 8:54:34 AM, SMS.TransactionCode: 88, SMS.Area: 0, SMS.Device: 3, SMS.Cardholder: -1) (Acknowledged: False, Secured:
True, AlarmCriteriaID: 2)
Received Alarm Alarm for Transaction: 1539 (4/1/2013 8:54:35 AM, SMS.TransactionCode: 96, SMS.Area: 0, SMS.Device: 3, SMS.Cardholder: -1) (Acknowledged: False, Secured:
True, AlarmCriteriaID: 2)
Received Alarm Alarm for Transaction: 1540 (4/1/2013 8:54:39 AM, SMS.TransactionCode: 96, SMS.Area: 0, SMS.Device: 3, SMS.Cardholder: -1) (Acknowledged: False, Secured:
```

- **Lookup Device Info for Real Time Transaction** - When this feature is enabled the real time transactions shown in the Logging window will also include information on the device where the transaction was generated.
- **Clear Log** - Clicking this clears the Logging window of all currently displayed information.

Read Database

The Read Database section allows the user to send queries to the SMS system. The information found is displayed in the Logging window.



[● Return Transactions] Return Alarms

Start Date/Time 2009-01-01T00:00:00.000

End Date/Time 2009-01-07T00:00:00.000

CardholderID -99

AreaID -99

DeviceID -99

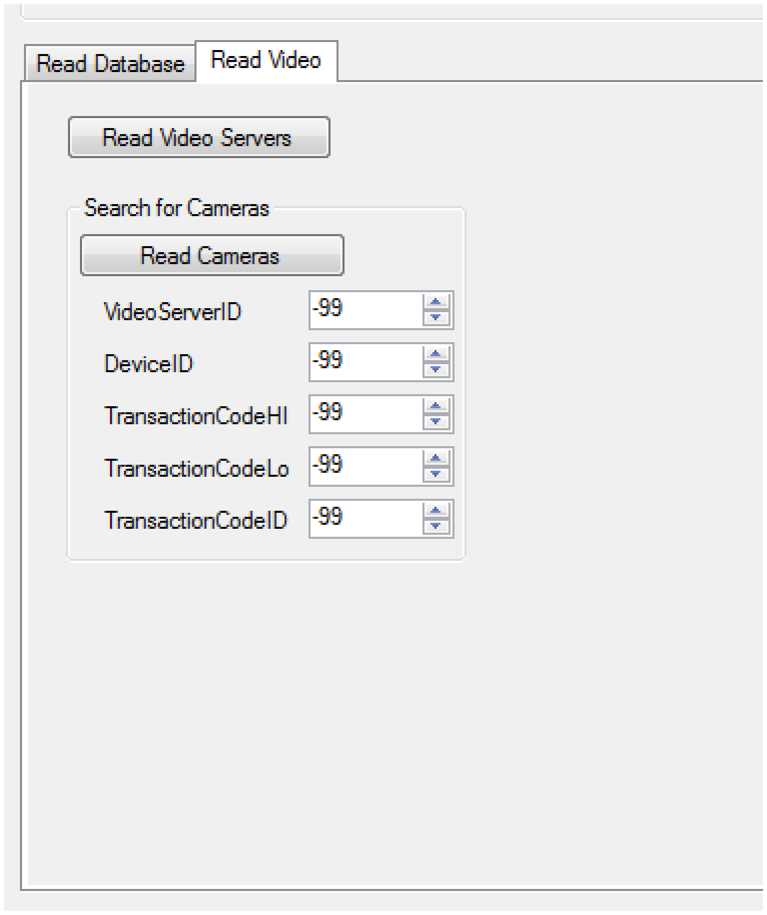
- **Read Options (unsorted)** - Contains the functions for reading MROs, MRO Sets, Areas, Area Sets, Devices, Device Types, Area Accessories, Transaction Codes, Transaction Groups and Deleted Devices.
- **Search for Cardholder** - Finds Cardholder information. Define the fields and click the Read Cardholders button.
- **Search for Transaction** - Searches for specific transactions or alarms. Define the fields and click the Read Transactions button.

Note: the value of **-99** in the **CardholderID**, **AreaID** and **DeviceID** fields denotes a "find all" function.

- **Cardholder Portrait Lookup** - Shows cardholder images. Use the drop down to specify which portrait to display. The portrait will display in the Cardholder Portrait Lookup section.
- **Alarm Comments** - Displays or inserts comments into the system.
- **Alarm Instructions** - Displays Alarm Criteria. Use the drop down to specify which Alarm Criteria.
- **Alarm Criteria** - Displays the Alarm Criteria Details. Use the drop down to specify which Alarm Criteria.

Read Video

The Read Video section allows the user to send queries to the SMS database regarding installed Video Servers. The information found is displayed in the Logging window.



VideoServerID	-99
DeviceID	-99
TransactionCodeHI	-99
TransactionCodeLo	-99
TransactionCodeID	-99

- **Read Video Servers** - Queries the SMS database for installed Video Servers.
- **Search for Cameras** - This section returns information on the Cameras connected to the designated SMS Video Server.

Note: the value of **-99** in the drop down fields denotes a "find all" function.

Index

VANDERBILT
INDUSTRIES

vanderbiltindustries.com

06021 12/12