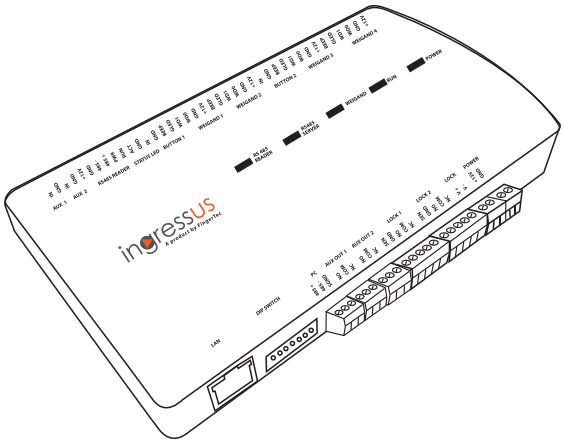
FINGERTEC



ingressUS Controllers

Centralization of Access Control Devices Made Easy

Ingressus SDK

Describe Ingressus SDK function

CONTENTS

• Ingressus SDK Introduction

The FingerTec Ingressus SDK manual shows developers how quickly and easily FingerTec Ingressus SDK can be used as a mean of data communication between applications and Ingressus.

What's in this Guide

Followings are the descriptions of each of the chapter

- How to install FingerTec Ingressus SDK
- Describes each of the function available in the FingerTec Ingressus SDK
- FingerTec Ingressus SDK Appendix

Requisite Knowledge

In order to use the guide successfully, you should have:

- Ability to write code and implement dll (dynamic link library) for the Windows application
- Installed with .NET framework 4.0
- Familiar with .NET languages such as C# and VB.Net.

Support Resources

In addition to this guide, the following resources are provided for additional support:

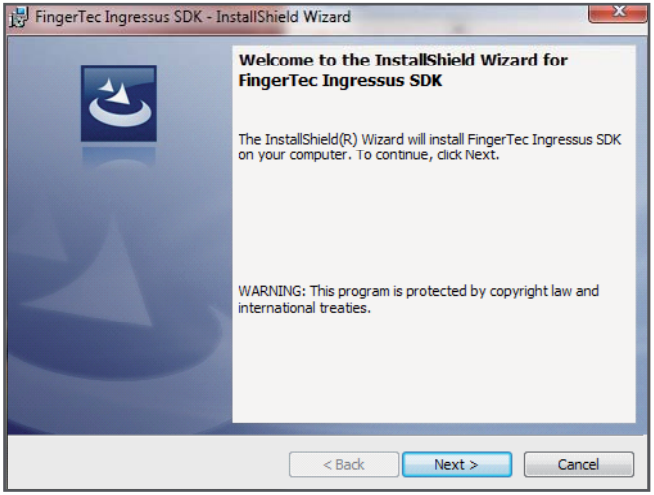
- This FingerTec Ingressus SDK manual is provided for developers. It has a list of descriptions and explanations of the SDK, the implementations and integrations of its functions into their existing application or systems.
- E-mail support is available at sdk@fingertec.com

Your Feedback Requested

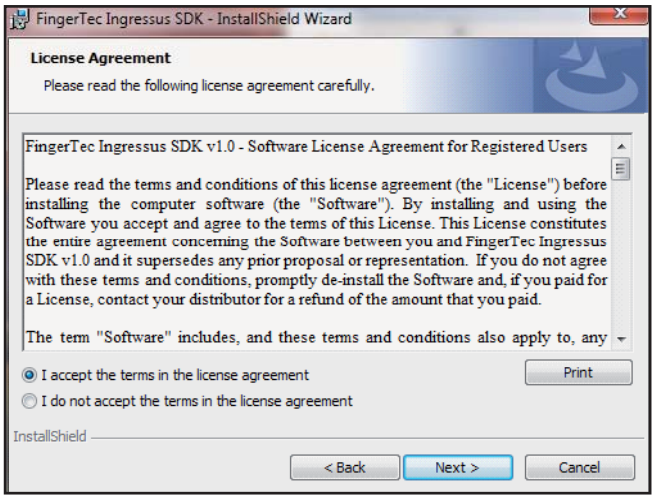
The information in this guide has been thoroughly reviewed and tested. If you find errors or have suggestions for future publications, contact *sdk@fingertec.com*

• FingerTec Ingressus SDK Installation

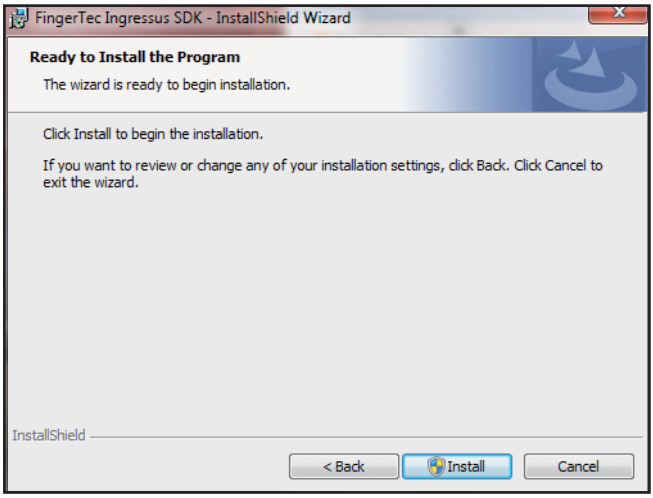
1. Double click on the icon FingerTec Ingressus SDK Setup.exe. A Window will pop up and click Next to start the installation.



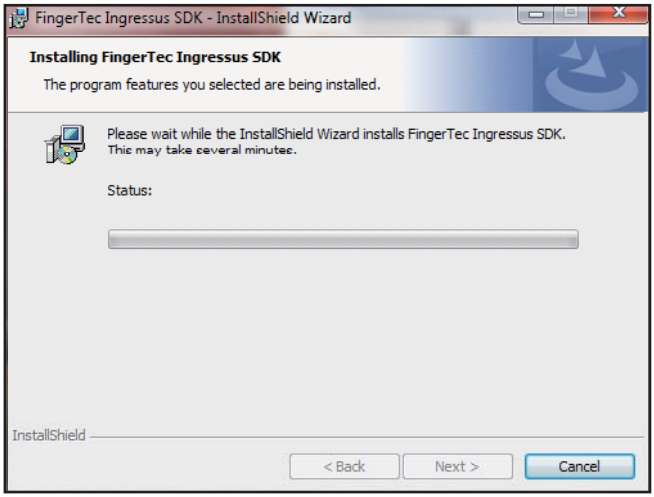
2. Please read the license agreement. You need to accept the license agreement to proceed to the installation process.



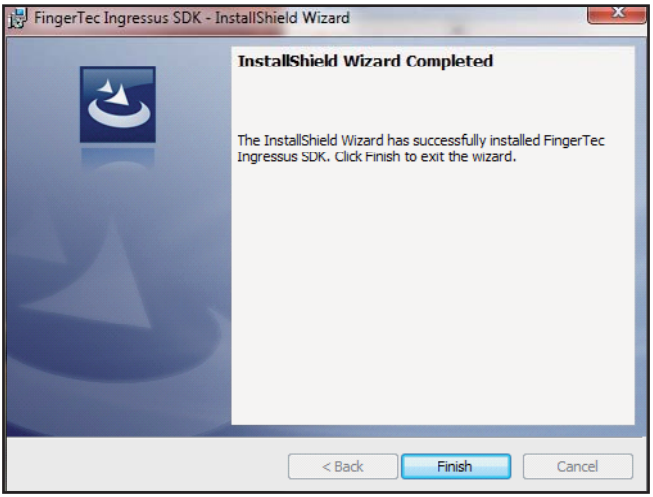
3. Enter your information in the fields provided.



4. The installation process is currently running. Please wait until it's finished.



5. The installation of the FingerTec Ingressus SDK is complete. Click Finish to exit the setup wizard.



• Ingressus SDK Functions

Device Control Management / Operation

Connect_TCPIP()

[Method]
Connect_TCPIP(string IPAddress, int Port)

[Feature]
Connect to a device using TCP/IP protocol

[Parameter Declaration]
Ipaddress = Device IP addresses (e.g. 192.168.1.210)
Port = Port number (default 4370)

[Return Value]
Return true for success, false for fail

[Example]

```
string ipaddress = "192.168.1.201";
int port = 4370;
if (sdk.Connect_TCPIP(ipaddress , port))
{
    MesssageBox.Show("Success");
}
else
{
    MesssageBox.Show ("Failed");
}
```

Connect_COMM()

[Method]
Connect_COMM(int ComPort , int Baudrate, int DeviceID)

[Feature]
Connect to a device using COMM Port

[Parameter Declaration]
ComPort = COMM port number. (e.g. 1,2,3)
Baudrate = Communcation baudrate. (e.g. 115200)
DeviceID = Device Number (e.g. 1,2,3)

[Return Value]
Return true for success, false for fail

[Example]
```
int CommPort = 1;
int baudrate = 115200;
int deviceID = 1;
      if (sdk. Connect_COMM (CommPort, baudrate, deviceID))
       {
              MesssageBox.Show("Success");
       }
       else
       {
              MesssageBox.Show ("Failed ");
       }
```

Disconnect ()

[Method]
Disconnect (void)

[Feature]
To disconnect from a connected device.

[Return Value]
Return true for success, false for fail

[Example]
```
 if (sdk. Disconnect())
{
      MesssageBox.Show("Success");
}
 else
{
      MesssageBox.Show ("Failed");
}
```

SetCommKey ()

[Method]
SetCommKey(string CommKey)

[Feature]
Set device communication key for authorized access.

[Parameter Declaration]
CommKey = Communication key. (e.g. 1, 2, 3)

[Return Value]
Return true for success, false for fail

[Example]

```
String CommKey = "123";
 if (sdk. SetCommKey(CommKey))
{
    MesssageBox.Show("Done ");
 }
else
{
    MesssageBox.Show ("Failed");
 }
```

SetConnectionTimeout()

[Method]
SetConnectionTimeout (int time)

[Feature]
Set device connection timeout.

[Parameter Declaration]
Time = Connection time in seconds (e.g. 1, 2, 3);
[Return Value]
Return true for success, false for fail

[Example]

```
int time = 5;
if (sdk. SetConnectionTimeout (time))
{
    MesssageBox.Show("Done ");
}
else
{
MesssageBox.Show ("Failed");
 }
```

RemoteOpenDoor()

[Method]
RemoteOpenDoor (int doorNo)

[Feature]
Remote opening door

[Parameter Declaration]
doorNo = Door number of Ingressus (e.g. 1,2);
[Return Value]
Return true for success, false for fail

[Example]

```
int doorNo = 1;
if (sdk. RemoteOpenDoor (doorNo))
{
    MesssageBox.Show(“Success “);
}
else
{
    MesssageBox.Show (“Failed”);
}
```

RemoteCloseDoor()

[Method]
RemoteCloseDoor (int doorNo)

[Feature]
Remote close door

[Parameter Declaration]
doorNo = Door number of Ingressus (e.g. 1,2);
[Return Value]
Return true for success, false for fail

[Example]

```
int doorNo = 1;
if (sdk. RemoteCloseDoor (doorNo))
{
    MesssageBox.Show(“Success “);
}
else
{
    MesssageBox.Show (“Failed”);
}
```

TurnOffAlarm()

[Method]
TurnOffAlarm (void)

[Feature]
To remote stop alarm.

[Return Value]
Return true for success, false for fail

[Example]

```
if (sdk. TurnOffAlarm ())
{
    MesssageBox.Show(“Success “);
}
else
{
    MesssageBox.Show (“Failed”);
}
```

RestartDevice()

[Method]
RestartDevice (void)

[Feature]
To restart Ingressus.

[Return Value]
Return true for success, false for fail

[Example]

```
if (sdk. RestartDevice ())
{
    MesssageBox.Show(“Success “);
}
else
{
    MesssageBox.Show (“Failed”);
}
```

SetDeviceTimeLocal()

[Method]
SetDeviceTimeLocal (void)

[Feature]
Set device date and time using PC local time

[Return Value]
Return true for success, false for fail

[Example]

```
if (sdk. SetDeviceTimeLocal ())
{
    MesssageBox.Show(“Success “);
}
else
{
    MesssageBox.Show (“Failed”);
}
```

SetDeviceTime()

[Method]
SetDeviceTimeLocal (int year, int month, int day, int hour, int minute, int second)

[Feature]
Set device date and time

[Parameter Declaration]
Year = year of date
Month = month of date
Day = day of date
Hour = hour of time
Minute = minute of time
Second = second of time

[Return Value]
Return true for success, false for fail

[Example]
int year = 2013;
int month = 1;
int day = 1;
int hour = 12;
int minute = 0;
int second = 59;
if (sdk. SetDeviceTime (year, month, day, hour, minute, second))
{
    MesssageBox.Show(“Success “);
}
else
{
    MesssageBox.Show (“Failed”);
}

GetDoorOptions()

[Method]
GetDoorOptions (int doorNo, string parameter)

[Feature]
Get Ingressus’s door options

[Parameter Declaration]

doorNo = Door number of Ingressus (e.g. 1,2);
parameter = The Ingresuss parameter to be get
* please refer to the appendix – Ingressus SDK Parameter

[Return Value]
Return door options

[Example]
int doorNo = 1;
string parameter = “SensorType”;
string value = sdk. GetDoorOptions (doorNo, parameter);

SetDoorOptions()

[Method]
SetDoorOptions (int doorNo, string parameter, string value)

[Feature]
Set Ingressus's door options

[Parameter Declaration]

doorNo = Door number of Ingressus (e.g. 1,2);
parameter = The Ingresuss parameter to be set
value = The Ingresuss parameter value to be set
* please refer to the appendix – Ingressus SDK Parameter

[Return Value]
Return true for success, false for fail

[Example]

```
int doorNo = 1;
string parameter = "SensorType";
string value = "0";
if( sdk. SetDoorOptions (doorNo, parameter, value))
{
    MessageBox.Show("Success");
}
else
{
    MessageBox.Show("Failed");
}
```

GetSysOptions()

[Method]
GetSysOptions (string parameter)

[Feature]
Get Ingressus's system options

[Parameter Declaration]

parameter = The Ingresuss parameter to be get
* please refer to the appendix – Ingressus SDK Parameter

[Return Value]
Return system options

[Example]
string parameter = “IPAddress”;
string value = sdk. GetSysOptions(parameter);

SetSysOptions()

[Method]
SetSysOptions (string parameter, string value)

[Feature]
Set Ingressus’s system options

[Parameter Declaration]
parameter = The Ingresuss parameter to be set
value = The Ingresuss parameter value to be set
* please refer to the appendix – Ingressus SDK Parameter

[Return Value]
Return true for success, false for fail

[Example]

```
string parameter = “IPAddress”;
string value = “192.168.1.201”;
if( sdk. SetSysOptions (parameter, value))
{
    MessageBox.Show(“Success”);
}
else
{
    MessageBox.Show(“Failed”);
}
```

Real Time Event Management

When you resolve single records, make adjustments according to Param 4 of the separated data. If Param 4 is 255, this record contains the door status and alarm status only; otherwise, this record contains realtime event records.

The following table compares the data structures of these two records.

	Date& Time	Param 1	Param 2	Param 3	Param 4	Param 5	Param 6
Door/ Alarm Status	Time	DSS status (0: no DSS; 1: door closed; 2: door opening timeout)	Alarm status (1: alarm; 2: door opening timeout)	Temporarily not in use	255	Temporarily not in use	200 (Indicates that the verification mode is "none"); not in use
Realtime Event Records	Time	UserID (Employee No.)	Card Num.	Door No., namely lock number	Event type code. See Attachment 3 for details.	Entry/ Exit status: (0: entry; 1: exit: 2: none)	The verification mode is the same as the door opening mode of controller parameters described in Attachment 1. 1:Fingerprint 4: Card 6:Card or fingerprint 10:Card and fingerprint 11: Card and password

Note:

(1) The Ingressus can temporarily save a maximum of 30 realtime event records. You can call GetRealTimeLog to check whether the cache contains event records. If so, the Ingressus returns all records (30 entries at most) in the current cache; otherwise, the device returns the door and alarm status events referred above.

(2) The door status records contain the open/closed status of current door (on the premise that the DSS is connected). Additionally, you can judge the current door status through "Door already open" (Event code: 200) and "Door already closed" (Event code: 201).

(3) When the record adopts "realtime event" status and type of event is Triggered Linkage Event (the code of type event: 6), the sixth place saved Linkage Event Triggered, and the second is for reuse of Linkage ID, It have software for the device synchronous linkage setting (usually the linkage in the ID value of software end database).

RealRealTimeLog()

[Method]
ReadRealTimeLog ()

[Feature]
Read and store all real time event log into memory buffer. To be retrieved by GetReal-TimeLog()

[Return Value]
Return true for success, false for fail

[Example]

```
if (sdk. ReadRealTimeLog())
{
        MesssageBox.Show("Success ");
}
else
{
        MesssageBox.Show ("Failed");
}
```

GetRealTimeLog()

[Method]
GetRealTimeLog (out string datetime, out string param1, out string param2, out string param3, out string param4, out string param5, out string param6)

[Feature]
Get a user record from memory buffer by ReadRealTimeLog()

[Parameter Declaration]

datetime = date and time of event
param1 = userid / Door sensor
param2 = card number / alarm status
param3 = door number
param4 = event type
param5 = Entry / Exit status
param6 = Verification mode
*Please refer to the appendix Event Type and Code for the event type description

[Return Value]
Return true for success, false for fail

[Example]
string datetime = string.Empty;
string param1 = string.Empty;
string param2 = string.Empty;
string param3 = string.Empty;
string param4 = string.Empty;
string param5 = string.Empty;
string param6 = string.Empty;

while (sdk.GetRealTimeLog(out datetime, out param1, out param2, out param3, out param4, out param5, out param6))
{
}

Device Data Management

RealAllDeviceData()

[Method]
RealAllDeviceData (string tableName)

[Feature]
Read and store all device data into memory buffer. To be retrieved by GetFirstCardData (), GetHolidayData(), GetTimezoneData(), GetUserInfo(), GetUserAccessLevel(), GetUserGeneralLog(), GetUserTemplate, GetFireAlarmData(), GetMultiCardData()

[Parameter Declaration]

tableName = name of table to be retrived

[Return Value]
Return true for success, false for fail

[Example]
String tableName = "user";
if (sdk. RealAllDeviceData (tableName))
{
MesssageBox.Show("Success ");
}
else
{
MesssageBox.Show ("Failed");
}

RealDeviceData()

[Method]
RealAllDeviceData (string tableName)

[Feature]
Read and store device data into memory buffer. To be retrieved by GetFirstCardData (), GetHolidayData(), GetTimezoneData(), GetUserInfo(), GetUserAccessLevel(), GetUserGeneralLog(), GetUserTemplate, GetFireAlarmData(), GetMultiCardData()

[Parameter Declaration]

tableName = name of table to be retrived

[Return Value]
Return true for success, false for fail

[Example]

```
String tableName = "user";
String userid = "1001";
if (sdk. RealAllDeviceData (tableName, userid))
{
        MesssageBox.Show("Success ");
}
else
{
        MesssageBox.Show ("Failed");
}
```

GetFirstCardData()

[Method]
GetFirstCardData (out int userid, out int doorNo, out int timezoneid)

[Feature]
Get first card data from memory buffer by RealDeviceData () / RealAllDeviceData()

[Parameter Declaration]

Userid = enrollment number
doorNo = Door number of Ingressus
Timezoneid = ID of timezone

[Return Value]
Return true for success, false for fail

[Example]
int userid = 0;
int doorNo = 0;
int timezoneid = 0;

while (sdk. GetFirstCardData (out userid, out doorNo, out timezoneid))
{
}

GetHolidayData()

[Method]
GetHolidayData (out string Holiday, out int HolidayType, out int Loop)

[Feature]
Get holiday data from memory buffer by RealDeviceData () / RealAllDeviceData()

[Parameter Declaration]

Holiday = Holiday date (Format : yyyyMMdd)
HolidayType = The HolidayType value can be 1, 2, and 3.
Loop = 1 (loop by year), 2 (not loop by year)
[Return Value]
Return true for success, false for fail

[Example]
string Holiday = 0;
int HolidayType = 0;
int Loop = 0;

while (sdk. GetHolidayData (out Holiday, out HolidayType, out Loop))
{
}

GetTimezoneData()

[Method]
GetTimezoneData (out long TimezoneId, out long SunTime1, out long SunTime2, out long SunTime3, out long MonTime1, out long MonTime2, out long MonTime3, out long TueTime1, out long TueTime2, out long TueTime3, out long WedTime1, out long WedTime2, out long WedTime3, out long ThuTime1, out long ThuTime2, out long ThuTime3, out long FriTime1, out long FriTime2, out long FriTime3, out long SatTime1, out long SatTime2, out long SatTime3, out long Hol1Time1, out long Hol1Time2, out long Hol1Time3, out long Hol2Time1, out long Hol2Time2, out long Hol2Time3, out long Hol3Time1, out long Hol3Time2, out long Hol3Time3)

[Feature]
Get timezone data from memory buffer by RealDeviceData () / RealAllDeviceData()

[Parameter Declaration]

TimezoneId = ID of timezone
SunTime1 = Time interval in hexadecimal
SunTime2 = Time interval in hexadecimal
SunTime3 = Time interval in hexadecimal
MonTime1 = Time interval in hexadecimal
MonTime2 = Time interval in hexadecimal
MonTime3 = Time interval in hexadecimal
TueTime1 = Time interval in hexadecimal
TueTime2 = Time interval in hexadecimal
TueTime3 = Time interval in hexadecimal
WedTime1 = Time interval in hexadecimal
WedTime2 = Time interval in hexadecimal
WedTime3 = Time interval in hexadecimal
ThuTime1 = Time interval in hexadecimal
ThuTime2 = Time interval in hexadecimal
ThuTime3 = Time interval in hexadecimal
FriTime1 = Time interval in hexadecimal
FriTime2 = Time interval in hexadecimal
FriTime3 = Time interval in hexadecimal
SatTime1 = Time interval in hexadecimal
SatTime2 = Time interval in hexadecimal
SatTime3 = Time interval in hexadecimal
Hol1Time1 = Time interval in hexadecimal
Hol1Time2 = Time interval in hexadecimal
Hol1Time3 = Time interval in hexadecimal
Hol2Time1 = Time interval in hexadecimal
Hol2Time2 = Time interval in hexadecimal
Hol2Time3 = Time interval in hexadecimal
Hol3Time1 = Time interval in hexadecimal
Hol3Time2 = Time interval in hexadecimal
Hol3Time3 = Time interval in hexadecimal

[Return Value]
Return true for success, false for fail
[Example]

```
int TimezoneId =0;
long SunTime1 = 0;
        long SunTime2 = 0;
        long SunTime3 = 0;
        long MonTime1 = 0;
        long MonTime2 = 0;
```

```
long MonTime3 = 0;
long TueTime1 = 0;
long TueTime2 = 0;
long TueTime3 = 0;
long WedTime1 = 0;
long WedTime2 = 0;
long WedTime3 = 0;
long ThuTime1 = 0;
long ThuTime2 = 0;
long ThuTime3 = 0;
long FriTime1 = 0;
long FriTime2 = 0;
long FriTime3 = 0;
long SatTime1 = 0;
long SatTime2 = 0;
long SatTime3 = 0;
long Hol1Time1 = 0;
long Hol1Time2 = 0;
long Hol1Time3 = 0;
long Hol2Time1 = 0;
long Hol2Time2 = 0;
long Hol2Time3 = 0;
long Hol3Time1 = 0;
long Hol3Time2 = 0;
long Hol3Time3 = 0;
```

while (sdk. GetTimezoneData(out TimezoneId, out SunTime1, out SunTime2, out SunTime3, out MonTime1, out MonTime2, out MonTime3, out TueTime1, out TueTime2, out TueTime3, out WedTime1, out WedTime2, out WedTime3, out ThuTime1, out ThuTime2, out ThuTime3, out FriTime1, out FriTime2, out FriTime3, out SatTime1, out SatTime2, out SatTime3, out Hol1Time1, out Hol1Time2, out Hol1Time3, out Hol2Time1, out Hol2Time2, out Hol2Time3, out Hol3Time1, out Hol3Time2, out Hol3Time3)
{
}

GetUserInfo()

[Method]
GetUserInfo (out int userID, out int privilege, out string username, out string cardNumber, out int multicardGroup, out string password, out string IssueTime, out string ExpireTime)

[Feature]
Get user data from memory buffer by RealDeviceData () / RealAllDeviceData()

[Parameter Declaration]

userID = Enrollment number
privilege = Privilege level
username = User name of user
cardNumber = Card number of user
multicardGroup = Group of multicard verification
password = Password of user
IssueTime = The user starting work period
ExpireTime = The user last work period

[Return Value]
Return true for success, false for fail

[Example]
```
int userID = 0;
int privilege = 0;
string username = string.Empty;
string cardNumber = string.Empty;
int  multicardGroup = 0;
string  password = string.Empty;
string  IssueTime = string.Empty;
string  ExpireTime = string.Empty;

while (sdk.  GetUserInfo(out userID, out privilege, out username, out cardNumber, out multicardGroup, out password, out IssueTime, out ExpireTime)
{
}
```

GetUserAccessLevel()

[Method]
GetUserAccessLevel(out int UserID, out int AuthorizeTimezoneId, out int AuthorizeDoorId)

[Feature]
Get user access level from memory buffer by RealDeviceData () / RealAllDeviceData()

[Parameter Declaration]

UserID = Enrollment number
AuthorizeTimezoneId = Authorize user ID
AuthorizeDoorId = Authorize door no

[Return Value]
Return true for success, false for fail

[Example]
int userID = 0;
 int AuthorizeTimezoneId = 0;
int AuthorizeDoorId = 0;

while (sdk. GetUserAccessLevel (out userID, out AuthorizeTimezoneId, out AuthorizeDoor-Id)
{
}

GetUserGeneralLog()

[Method]
GetUserGeneralLog (out int UserID, out string CardNumber, out int Verified, out int Door-No, out int EventType, out int InOutState, out int TimeSecond)

[Feature]
Get user general log from memory buffer by RealDeviceData () / RealAllDeviceData()

[Parameter Declaration]

UserID = Enrollment number
CardNumber = Card number of user
Verified = The Verified mode can be as follows:
    1:Only finger
    3: Only password
    4: Only card
    11: Card and password
    200: Others
DoorNo = Transaction trigger door No
EventType = Event type of this record
InOutState = In and Out Mode
TimeSecond = TimeSecond should be specified in a correct format: YYYY-MM-DD hh:mm:ss
    (After writing, data formats will conversion, if want to take out to analysis, analytical formula is as follows:
    second = t % 60;
    t /= 60;
    minute = t % 60;
    t /= 60;
    hour = t % 24;
    t /= 24;
    day = t % 31 + 1;
    t /= 31;
    month = t % 12 + 1;
    t /= 12;
        year = t + 2000);

[Return Value]
Return true for success, false for fail

[Example]
int userID = 0;
string CardNumber = string.Empty;
int Verified = 0;
int DoorNo = 0;
int EventType = 0;
int InOutState = 0;
int TimeSecond = 0;

while (sdk. GetUserGeneralLog (out UserID, out CardNumber, out Verified, out DoorNo, out EventType, out InOutState, out TimeSecond)
{
}

GetUserTemplate()

[Method]
GetUserTemplate (out int UserID, out int FingerID, out int Valid, out string Template)

[Feature]
Get user template from memory buffer by RealDeviceData () / RealAllDeviceData()

[Parameter Declaration]

UserID = Enrollment number
FingerID = ID of finger
Valid = Indicates a valid template (0 – Invalid , 1 – Valid)
Template = Template of finger

[Return Value]
Return true for success, false for fail

[Example]
int userID = 0;
int FingerID = 0;
int Valid = 0;
string Template = string.Empty;

while (sdk. GetUserTemplate (out userID, out FingerID, out Valid, out Template)
{
}

GetFireAlarmData ()

[Method]
GetFireAlarmData(out int Index, out int EventType, out int InAddr, out int OutType, out int OutAddr, out int OutTime)

[Feature]
Get fire alarm data from memory buffer by RealDeviceData () / RealAllDeviceData()

[Parameter Declaration]

Index = ID of fire alarm record
EventType = See Table 3 .
When the EventType value is 220 (the auxiliary input point is off) or 221 (the auxiliary input point is short-circuited), the input point is the auxiliary input. When the EventType value is not 220 or 221, the input point is a door.

InAddr = The input point InAddr is a door:
0: Any door
1: Door 1
2: Door 2
3: Door 3
4: Door 4
The input point is the auxiliary input:
0: Any auxiliary input
1: Auxiliary input 1
2: Auxiliary input 2
3: Auxiliary input 3
4: Auxiliary input 4

OutType = Indicate is lock or auxiliary (0 – Lock, 1 – auxiliary)
OutAddr = When the OutType value is 0, the output point OutAddr indicates a lock:
1: Lock 1
2: Lock 2
3: Lock 3
4: Lock 4

When the OutType value is 1, the output point OutAddr indicates the auxiliary output:
1: Auxiliary output 1
2: Auxiliary output 2
3: Auxiliary output 3
4: Auxiliary output 4
5: Auxiliary output 5
6: Auxiliary output 6

OutTime = Indicates lock / auxiliary open/close time

[Return Value]
Return true for success, false for fail

[Example]
int Index=0;
int EventType=0;
int InAddr=0;
int OutType=0;
int OutAddr=0;
int OutTime =0;

while (sdk. GetFireAlarmData (out Index, out EventType, out InAddr, out OutType, out Out-Addr, out OutTime)
{
}

GetMultiCardData ()

[Method]
GetMultiCardData(out int Index, out int DoorNo, out int Group1, out int Group2, out int Group3, out int Group4, out int Group5)

[Feature]
Get multi card data from memory buffer by RealDeviceData () / RealAllDeviceData()

[Parameter Declaration]

Index = ID of multicard record
DoorNo = Door no of Ingressus
Group1 = number of the multi-card opening groups
Group2 = number of the multi-card opening groups

Group3 = number of the multi-card opening groups
Group4 = number of the multi-card opening groups
Group5 = number of the multi-card opening groups

[Return Value]
Return true for success, false for fail

[Example]
int Index=0;
int DoorNo =0;
int Group1=0;
int Group2=0;
int Group3=0;
int Group4=0;
int Group5=0;

while (sdk GetMultiCardData(out Index, out DoorNo, out Group1, out Group2, out Group3, out Group4, out Group5)
{
}

ClearFirstCardData ()

[Method]
ClearFirstCardData ( )

[Feature]
Clear all first card data.

[Return Value]
Return true for success, false for fail

[Example]

```
if (sdk. ClearFirstCardData ())
{
    MessageBox.Show("Done");
}
else
{
    MessageBox.Show("Failed");
}
```

DeleteFirstCardData ()

[Method]
DeleteFirstCardData (int userid)

[Feature]
Delete user first card data.

[Parameter Declaration]
userid = Enrollment number

[Return Value]
Return true for success, false for fail

[Example]
Int userid = 1001;

```
if (sdk. DeleteFirstCardData (userid))
{
    MessageBox.Show("Done");
}
else
{
    MessageBox.Show("Failed");
}
```

ClearHolidayData ()

[Method]
ClearHolidayData ( )

[Feature]
Clear all holiday data.

[Return Value]
Return true for success, false for fail

[Example]

```
if (sdk. ClearHolidayData ())
{
    MessageBox.Show("Done");
}
else
{
    MessageBox.Show("Failed");
}
```

DeleteHolidayData ()

[Method]
DeleteHolidayData (string Holiday)

[Feature]
Delete holiday data.

[Parameter Declaration]
Holiday = Holiday date (Format: YYYYMMDD)

[Return Value]
Return true for success, false for fail

[Example]

```
String Holiday = "20130101";

if (sdk. DeleteHolidayData (Holiday))
{
 MessageBox.Show("Done");
}
else
{
MessageBox.Show("Failed");
}
```

ClearTimezoneData ()

[Method]
ClearTimezoneData ( )

[Feature]
Clear all timezone data.

[Return Value]
Return true for success, false for fail

[Example]

```
if (sdk. ClearTimezoneData ())
{
    MessageBox.Show("Done");
}
else
{
    MessageBox.Show("Failed");
}
```

DeleteTimezoneData ()

[Method]
DeleteTimezoneData (int TimezoneID)

[Feature]
Delete timezone data.

[Parameter Declaration]
TimezoneID = ID of timezone

[Return Value]
Return true for success, false for fail

[Example]

```
int TimezoneID = 1;

if (sdk. DeleteTimezoneData (TimezoneID))
{
    MessageBox.Show("Done");
}
else
{
    MessageBox.Show("Failed");
}
```

ClearFireAlarmData ()

[Method]
ClearFireAlarmData ( )

[Feature]
Clear all fire alarm data.

[Return Value]
Return true for success, false for fail

[Example]

```
if (sdk. ClearFireAlarmData ())
{
    MessageBox.Show("Done");
}
else
{
    MessageBox.Show("Failed");
}
```

DeleteFireAlarmData ()

[Method]
DeleteFireAlarmData (int index)

[Feature]
Delete fire alarm data.

[Parameter Declaration]
index = ID of fire alarm

[Return Value]
Return true for success, false for fail

[Example]

```
int index = 1;

if (sdk. DeleteFireAlarmData (index))
{
    MessageBox.Show("Done");
}
else
{
    MessageBox.Show("Failed");
}
```

ClearMultiCardData ()

[Method]
ClearMultiCardData ( )

[Feature]
Clear all multicard data data.

[Return Value]
Return true for success, false for fail

[Example]

```
if (sdk. ClearMultiCardData ())
{
    MessageBox.Show("Done");
}
else
{
    MessageBox.Show("Failed");
}
```

DeleteMultiCardData ()

[Method]
DeleteMultiCardData (int index)

[Feature]
Delete multi card data.

[Parameter Declaration]
index = ID of multi card

[Return Value]
Return true for success, false for fail

[Example]

```
int index = 1;

if (sdk. DeleteMultiCardData (index))
{
    MessageBox.Show("Done");
}
else
{
    MessageBox.Show("Failed");
}
```

ClearUserInfo ()

[Method]
ClearUserInfo ( )

[Feature]
Clear all enrolled user information.

[Return Value]
Return true for success, false for fail

[Example]

```
if (sdk. ClearUserInfo ())
{
    MessageBox.Show("Done");
}
else
{
    MessageBox.Show("Failed");
}
```

DeleteUserInfo ()

[Method]
DeleteUserInfo (int userid)

[Feature]
Delete enrolled user information.

[Parameter Declaration]
userid = Enrollment number

[Return Value]
Return true for success, false for fail

[Example]

```
int userid = 1001;

if (sdk. DeleteUserInfo (userid))
{
    MessageBox.Show("Done");
}
else
{
    MessageBox.Show("Failed");
}
```

ClearUserAccessLevel ()

[Method]
ClearUserAccessLevel ( )
[Feature]
Clear all enrolled user access level.

[Return Value]
Return true for success, false for fail

[Example]

```
if (sdk. ClearUserAccessLevel ())
{
    MessageBox.Show("Done");
}
else
{
    MessageBox.Show("Failed");
}
```

DeleteUserAccessLevel()

[Method]
DeleteUserAccessLevel (int userid)

[Feature]
Delete enrolled user access level.

[Parameter Declaration]
userid = Enrollment number

[Return Value]
Return true for success, false for fail

[Example]

```
int userid = 1001;

if (sdk. DeleteUserAccessLevel (userid))
{
    MessageBox.Show("Done");
}
else
{
    MessageBox.Show("Failed");
}
```

ClearUserGeneralLog()

[Method]
ClearUserGeneralLog ( )
[Feature]
Clear all user transaction logs.

[Return Value]
Return true for success, false for fail

[Example]

```
if (sdk. ClearUserGeneralLog ())
{
    MessageBox.Show("Done");
}
else
{
    MessageBox.Show("Failed");
}
```

DeleteUserGeneralLog()

[Method]
DeleteUserGeneralLog (int userid)

[Feature]
Delete user transaction log.

[Parameter Declaration]
userid = Enrollment number

[Return Value]
Return true for success, false for fail

[Example]

```
int userid = 1001;

if (sdk. DeleteUserGeneralLog (userid))
{
    MessageBox.Show("Done");
}
else
{
    MessageBox.Show("Failed");
}
```

ClearUserTemplate()

[Method]
ClearUserTemplate ( )

[Feature]
Clear all enrolled user template.

[Return Value]
Return true for success, false for fail

[Example]

```
if (sdk. ClearUserTemplate ())
{
    MessageBox.Show("Done");
}
else
{
    MessageBox.Show("Failed");
}
```

DeleteUserTemplate()

[Method]
DeleteUserGeneralLog (int userid)

[Feature]
Delete enrolled user template.

[Parameter Declaration]
userid = Enrollment number

[Return Value]
Return true for success, false for fail

[Example]

```
int userid = 1001;

if (sdk. DeleteUserGeneralLog (userid))
{
    MessageBox.Show("Done");
}
else
{
    MessageBox.Show("Failed");
}
```

ClearAllUserData()

[Method]
ClearAllUserData ( )

[Feature]
Clear all enrolled users' information, fingerprint templates, and access levels.

[Return Value]
Return true for success, false for fail

[Example]

```
if (sdk. ClearAllUserData ())
{
    MessageBox.Show("Done");
}
else
{
    MessageBox.Show("Failed");
}
```

DeleteUserData()

[Method]
DeleteUserData (int userid)

[Feature]
Delete enrolled users information, fingerprint templates, and access levels.

[Parameter Declaration]
userid = Enrollment number

[Return Value]
Return true for success, false for fail

[Example]

```
int userid = 1001;

if (sdk. DeleteUserData (userid))
{
    MessageBox.Show("Done");
}
else
{
    MessageBox.Show("Failed");
}
```

WriteAllDataToDevice()

[Method]
WriteAllDataToDevice ()

[Feature]
Read all memory buffer from SetFirstCardData () / SetHolidayData() / SetTimezoneData() / SetUserInfo() / SetUserAccessLevel() / SetUserGeneralLog() / SetUserTemplate() / SetFireAlarmData() / SetMultiCardData() and upload to Ingressus.

[Return Value]
Return true for success, false for fail

[Example]

```
 if (sdk. WriteAllDataToDevice ())
 {
     MessageBox.Show("Done");
}
else
{
     MessageBox.Show("Failed");
}
```

SetFirstCardData()

[Method]
SetFirstCardData (int userid, int doorNo, int timezoneid)

[Feature]
Write first card data to memory buffer.

[Parameter Declaration]

Userid = enrollment number
doorNo = Door number of Ingressus
Timezoneid = ID of timezone

[Return Value]
Return true for success, false for fail

[Example]

```
int userid = 1001;
int doorNo = 1;
int timezoneid = 1;
sdk. SetFirstCardData (userid, doorNo, timezoneid);
```

SetHolidayData()

[Method]
SetHolidayData (string Holiday, int HolidayType, int Loop)

[Feature]
Write holiday data to memory buffer.

[Parameter Declaration]

Holiday = Holiday date (Format: yyyyMMdd)
HolidayType = The HolidayType value can be 1, 2, and 3.
Loop = 1 (loop by year), 2 (not loop by year)
[Return Value]
Return true for success, false for fail

[Example]
```
 string Holiday = 20130101;
 int HolidayType = 1;
 int Loop = 1;

sdk. SetHolidayData (Holiday, HolidayType, Loop);
```

SetTimezoneData()

[Method]
SetTimezoneData(long TimezoneId, long SunTime1, long SunTime2, long SunTime3, long MonTime1, long MonTime2, long MonTime3, long TueTime1, long TueTime2, long TueTime3, long WedTime1, long WedTime2, long WedTime3, long ThuTime1, long ThuTime2, long ThuTime3, long FriTime1, long FriTime2, long FriTime3, long SatTime1, long SatTime2, long SatTime3, long Hol1Time1, long Hol1Time2, long Hol1Time3, long Hol2Time1, long Hol2Time2, long Hol2Time3, long Hol3Time1, long Hol3Time2, long Hol3Time3)

[Feature]
Set timezone data to memory buffer.

[Parameter Declaration]

TimezoneId = ID of timezone
SunTime1 = Time interval in hexadecimal
SunTime2 = Time interval in hexadecimal
SunTime3 = Time interval in hexadecimal
MonTime1 = Time interval in hexadecimal
MonTime2 = Time interval in hexadecimal
MonTime3 = Time interval in hexadecimal
TueTime1 = Time interval in hexadecimal

TueTime2 = Time interval in hexadecimal
TueTime3 = Time interval in hexadecimal
WedTime1 = Time interval in hexadecimal
WedTime2 = Time interval in hexadecimal
WedTime3 = Time interval in hexadecimal
ThuTime1 = Time interval in hexadecimal
ThuTime2 = Time interval in hexadecimal
ThuTime3 = Time interval in hexadecimal
FriTime1 = Time interval in hexadecimal
FriTime2 = Time interval in hexadecimal
FriTime3 = Time interval in hexadecimal
SatTime1 = Time interval in hexadecimal
SatTime2 = Time interval in hexadecimal
SatTime3 = Time interval in hexadecimal
Hol1Time1 = Time interval in hexadecimal
Hol1Time2 = Time interval in hexadecimal
Hol1Time3 = Time interval in hexadecimal
Hol2Time1 = Time interval in hexadecimal
Hol2Time2 = Time interval in hexadecimal
Hol2Time3 = Time interval in hexadecimal
Hol3Time1 = Time interval in hexadecimal
Hol3Time2 = Time interval in hexadecimal
Hol3Time3 = Time interval in hexadecimal

[Return Value]
Return true for success, false for fail
[Example]

```
int TimezoneId =1;
long SunTime1 =2359; // full access, from 12am to 1159pm
long SunTime2 =0;
long SunTime3 =0;
long MonTime1 =2359;
long MonTime2 =0;
long MonTime3 =0;
long TueTime1 =2359;
long TueTime2 =0;
long TueTime3 =0;
long WedTime1 =2359;
long WedTime2 =0;
long WedTime3 =0;
long ThuTime1 =2359;
long ThuTime2 =0;
long ThuTime3 =0;
long FriTime1 =2359;
long FriTime2 =0;
long FriTime3 =0;
long SatTime1 =2359;
```

long SatTime2 =0;
long SatTime3 =0;
long Hol1Time1 =2359;
long Hol1Time2 =0;
long Hol1Time3 =0;
long Hol2Time1 =2359;
long Hol2Time2 =0;
long Hol2Time3 =0;
long Hol3Time1 =2359;
long Hol3Time2 =0;
long Hol3Time3 =0;

sdk. SetTimezoneData(TimezoneId, SunTime1, SunTime2, SunTime3, MonTime2, MonTime3, TueTime1, TueTime2, TueTime3, WedTime1, WedTime2,WedTime3, ThuTime1, ThuTime2, ThuTime3, FriTime1, FriTime2, FriTime3, SatTime1, SatTime2, SatTime3, Hol1Time1, Hol1Time2, Hol1Time3, Hol2Time1, Hol2Time2, Hol2Time3, Hol3Time1, Hol3Time2, Hol3Time3);

SetUserInfo()

[Method]
SetUserInfo (int userID, int privilege, string username, string cardNumber, int multicardGroup, string password, string IssueTime, string ExpireTime)

[Feature]
Write user data to memory buffer.

[Parameter Declaration]

userID = Enrollment number
privilege = Privilege level
username = User name of user
cardNumber = Card number of user
multicardGroup = Group of multicard verification
password = Password of user
IssueTime = The user starting work period
ExpireTime = The user last work period

[Return Value]
Return true for success, false for fail

[Example]
int userID =1001;
int privilege = 1;
string username = "George";
string cardNumber = "9733707";

```
int  multicardGroup = 0;
string  password =”123”;
string  IssueTime = ” 20130101”;
string  ExpireTime = ”20131231”;

sdk.SetUserInfo(userID, privilege, username, cardNumber, multicardGroup, password, Is-
sueTime, ExpireTime);
```

SetUserAccessLevel()

[Method]
SetUserAccessLevel(int UserID, int AuthorizeTimezoneId, int AuthorizeDoorNo)

[Feature]
Write user access level to memory buffer.

[Parameter Declaration]

UserID = Enrollment number
AuthorizeTimezoneId = Authorize user ID
AuthorizeDoorNo = Authorize door no

[Return Value]
Return true for success, false for fail

[Example]

```
int userID = 1001;
int AuthorizeTimezoneId = 1;
int  AuthorizeDoorNo = 1;

sdk.  GetUserAccessLevel (userID, AuthorizeTimezoneId, AuthorizeDoorNo);
```

SetUserGeneralLog()

[Method]
SetUserGeneralLog (int UserID, string CardNumber, int Verified, int DoorNo
, int EventType, int InOutState, int TimeSecond)

[Feature]
Write user general log to memory buffer.

[Parameter Declaration]

UserID = Enrollment number
CardNumber = Card number of user

Verified = The Verified mode can be as follows:
1: Only finger
3: Only password
4: Only card
11: Card and password
200: Others
DoorNo = Transaction trigger door No
EventType = Event type of this record
InOutState = In and Out Mode
TimeSecond = TimeSecond should be specified in a correct format: YYYY-MM-DD hh:mm:ss
(After writing, data formats will conversion, if want to take out to analysis, analytical formula is as follows:
second = t % 60;
t /= 60;
minute = t % 60;
t /= 60;
hour = t % 24;
t /= 24;
day = t % 31 + 1;
t /= 31;
month = t % 12 + 1;
t /= 12;
year = t + 2000);

[Return Value]
Return true for success, false for fail

[Example]
```
int userID = 1001;
string CardNumber = "9733707";
int  Verified = 200;
int  DoorNo = 1;
int  EventType = 0;
int  InOutState = 1;
int  TimeSecond = 418657364;

sdk.SetUserGeneralLog (UserID, CardNumber, Verified, DoorNo, EventType, InOutState, TimeSecond );
```

SetUserTemplate()

[Method]
SetUserTemplate (int UserID, int FingerID, int Valid, string Template)

[Feature]
Write user template to memory buffer.

[Parameter Declaration]

UserID = Enrollment number
FingerID = ID of finger
Valid = Indicates a valid template (0 – Invalid , 1 – Valid)
Template = Template of finger

[Return Value]
Return true for success, false for fail

[Example]

```
int userID = 1001;
 int FingerID = 1;
int  Valid = 1;
string Template = "SrNTUzIxAAAD8PUECAUHCc7QAAAb8WkBAAAAgx0gcfAsA-
HYPdgCCAHf/lwBHAA0PaQBV8IgPiwBjAM4PjfBuABAPYwBDAFz/
LACWAFMPXwCb8BQO5gCtAGcPgfCuAFIPxwByAC7/ZgC4AEcPRgC48EIP-
MwC8AI4PtfC8ADYPWQAbAD3/tADfAD8P7ADn8MgPqADzAIIPgPD8AL4NLADFATj9P-
gALATkKQgAP8cEDQAAfAX4KuvAiAVAN2ADmAUj2MAArAbYNWAAv8WoO4gBP
ARYBnPBRAdsOQgCTAar+RAcmaztvAAMNmDNpjYfGBDOYko9rgV56rP+DAB1pKQdWij-
oGylBc6UoNagoj9oIPzTTH1yIHUnSWbTyXbROhDxI8APIJfKfzPQct908K+wPA/
xYz8vpHhlT9PAmO+fMg1/q8hOL1mfoGCdMX9mODguaTEQp3iqF6+AC6el+BmAYCh2
+BhYHdAl+FdvTrVqZ6KQVAhW1xpOcyBOcJc/xti7Ljsf2efUN/Cu9Xkb6uZouWhywLZ3F/
jacLTvSUApZzhgZrsOU0AvJPIRwGAJ7vhncyAgBuLnfBzgC+zIj/d397CcVyR4RoeMEHAJ
mABizfDQByS3RvscJm+AGgTQw2//sRA1NVjJZvwcG9wF/7AY5jCf cHBQcMX8DGS2sD+/
zjAww3A/P////86wPww/yYAopaWBMWjeHnBev6WwwXDnDTAw2p4aBrF85ZXf8F5/
8KDBMHBMcFwZ8EKAOyZVZTC/2wTAJ1cHj3C/0RL/
v4tyQAsalLBWsFxessAqmwfOz7AQf4HCgN2rB76wP5AqQoDj-
q9excGEwqAEAxmxJEMJAIJ2U8CFwMDABwCHcTc9yAoAybspNQU+wgwIAGW+SYuw
wADwM8BMwQUARsFFjg4AWdnD/znA/g34/Cr//f/mALYqu4DCwcPGAcLBMcLBZ8H-
BwgXCwjPDxsXDwcEHwGz8AVbhQ4uLAa8J8FrhPXh3yAARA0njPUb//zU8/vjD/gcApfdG/
hcH8K35RkEgAOn8wLxW/v/+/v46/P0PPlQzRT8E1SoHsIIJEC8EPbLC/GUhED8HvT6Kwf8-
PwfrA+/7/OsD8Mf///cD/wDrA/DH9ChBDDze/glDoEXkZq/5LOv/8DPv6+v39wDvA/TD///
nzBBDsJiqpBRC5KE8vwBAQxChrBRDLRpk0AOAXRSvCAxDPSi4yAhDRTFP8";
```

```
 sdk. SetUserTemplate (userID, FingerID, Valid, Template);
```

SetFireAlarmData ()

[Method]
SetFireAlarmData(int Index, int EventType, int InAddr, int OutType, int OutAddr, int OutTime)

[Feature]
Write fire alarm data to memory buffer.

[Parameter Declaration]

Index = ID of fire alarm record
EventType = See Table 3.
When the EventType value is 220 (the auxiliary input point is off) or 221 (the auxiliary input point is short-circuited), the input point is the auxiliary input. When the EventType value is not 220 or 221, the input point is a door.

InAddr = The input point InAddr is a door:
- 0: Any door
- 1: Door 1
- 2: Door 2
- 3: Door 3
- 4: Door 4

The input point is the auxiliary input:
- 0: Any auxiliary input
- 1: Auxiliary input 1
- 2: Auxiliary input 2
- 3: Auxiliary input 3
- 4: Auxiliary input 4

OutType = Indicate is lock or auxiliary (0 – Lock, 1 – auxiliary)
OutAddr = When the OutType value is 0, the output point OutAddr indicates a lock:
- 1: Lock 1
- 2: Lock 2
- 3: Lock 3
- 4: Lock 4

When the OutType value is 1, the output point OutAddr indicates the auxiliary output:
- 1: Auxiliary output 1
- 2: Auxiliary output 2
- 3: Auxiliary output 3
- 4: Auxiliary output 4
- 5: Auxiliary output 5
- 6: Auxiliary output 6

OutTime = Indicates lock / auxiliary open/close time

[Return Value]
Return true for success, false for fail

[Example]
int Index=1;
int EventType=0;
int InAddr=0;
int OutType=0;
int OutAddr=1;
int OutTime =20;

sdk. SetFireAlarmData (Index, EventType, InAddr, OutType, OutAddr, OutTime);

SetMultiCardData ()

[Method]
SetMultiCardData(int Index, int DoorNo, int Group1, int Group2, int Group3, int Group4, int Group5)

[Feature]
Write multi card data to memory buffer.
[Parameter Declaration]

Index = ID of multicard record
DoorNo = Door no of Ingressus
Group1 = number of the multi-card opening groups
Group2 = number of the multi-card opening groups
Group3 = number of the multi-card opening groups
Group4 = number of the multi-card opening groups
Group5 = number of the multi-card opening groups

[Return Value]
Return true for success, false for fail

[Example]
int Index=1;
int DoorNo =1;
int Group1=1;
int Group2=1;
int Group3=2;
int Group4=2;
int Group5=2;

sdk .GetMultiCardData(Index, DoorNo, Group1, Group2, Group3, Group4, Group5);

GetTotalUserCount ()

[Method]
GetTotalUserCount ()

[Feature]
Get total numbers of enrolled user.

[Return Value]
Return true for success, false for fail

[Example]
```
Int userCount =  sdk. GetTotalUserCount ();
```

GetTotalUserFPCount ()

[Method]
GetTotalUserFPCount ()

[Feature]
Get total numbers of enrolled user template.
[Return Value]
Return true for success, false for fail

[Example]
```
Int fpCount =  sdk. GetTotalUserFPCount ();
```

GetTotalGeneralLogCount ()

[Method]
GetTotalGeneralLogCount ()

[Feature]
Get total records of user transaction.

[Return Value]
Return true for success, false for fail

[Example]
```
Int transactionCount =  sdk. GetTotalGeneralLogCount ();
```

ScanDevice ()
[Method]
ScanDevice ()

[Feature]
Scan Ingressus in the network and store all information into memory bugger. To be retrieved by GetScanDevice().

[Return Value]
Return true for success, false for fail

[Example]
```
If(sdk.ScanDevice())
{
    MessageBox.Show("Done");
}
Else
{
    MessageBox.Show("Failed");
}
```

- GetScanDevice ()

[Method]
GetScanDevice (out string MAC, out string serialNumber, out string ipaddress, out string gateway, out string subnet, out string firmVer, out string model)

[Feature]
Get a device information from memory bugger by ScanDevice()

[Parameter Declaration]
MAC = MacAddress of Ingressus
serialNumber = serial number of Ingressus
ipaddress = IP Address of Ingressus
gateway = Gateway of Ingressus
subnet = Subnet mask of Ingressus
firmVer = Firmware version of Ingressus
model = Model of Ingressus

[Return Value]
Return true for success, false for fail

[Example]
```
String mac = string.Empty;
String serialNumber  = string.Empty;
String ipaddress = string.Empty;
String gateway  = string.Empty;
```

```
String subnet = string.Empty;
String firmVer = string.Empty;
String model = string.Empty;

While (sdk.GetScanDevice (out string MAC, out string serialNumber, out string ipaddress,
out string gateway, out string subnet, out string firmVer, out string model))
{
}
```

ChangeDeviceInfo ()

[Method]
ChangeDeviceInfo (string MAC, string ipaddress, string gateway, string subnet)

[Feature]
Directly change device information by knowing its MAC address

[Parameter Declaration]
MAC = MacAddress of Ingressus
ipaddress = IP Address of Ingressus that intent to change
gateway = Gateway of Ingressus that intent to change
subnet = Subnet mask of Ingressus that intent to change

[Return Value]
Return true for success, false for fail

[Example]

```
String mac = "00:17:61:10:29:BC";
String ipaddress = "192.168.1.201";
String gateway = "192.168.1.1";
String subnet ="255.255.255.0";

if(sdk.ChangeDeviceInfo (MAC, ipaddress, gateway ,subnet))
{
    MessageBox.Show("Success");
}
else
{
    MessageBox.Show("Failed");
}
```

ChangeDeviceIpAddress()

[Method]
ChangeDeviceIpAddress (string MAC, string ipaddress)

[Feature]
Directly change device IP Address by knowing its MAC address

[Parameter Declaration]
MAC = MacAddress of Ingressus
ipaddress = IP Address of Ingressus that intent to change

[Return Value]
Return true for success, false for fail

[Example]

```
String mac = "00:17:61:10:29:BC";
String ipaddress = "192.168.1.201";

if(sdk. ChangeDeviceIpAddress (MAC, ipaddress))
{
    MessageBox.Show("Success");
}
else
{
    MessageBox.Show("Failed");
}
```

ChangeDeviceGateway ()

[Method]
ChangeDeviceGateway (string MAC, string gateway)

[Feature]
Directly change device gateway by knowing its MAC address

[Parameter Declaration]
MAC = MacAddress of Ingressus
gateway = Gateway of Ingressus that intent to change

[Return Value]
Return true for success, false for fail

[Example]

```
String mac = "00:17:61:10:29:BC";
String gateway  = "192.168.1.1";
```

```
if(sdk. ChangeDeviceGateway (MAC, gateway ))
{
    MessageBox.Show("Success");
}
else
{
    MessageBox.Show("Failed");
}
```

ChangeDeviceSubnet ()

[Method]
ChangeDeviceSubnet (string MAC, string subnet)

[Feature]
Directly change device subnet mask by knowing its MAC address

[Parameter Declaration]
MAC = MacAddress of Ingressus
subnet = Subnet mask of Ingressus that intent to change

[Return Value]
Return true for success, false for fail

[Example]

```
String mac = "00:17:61:10:29:BC";
String subnet = "255.255.255.0";

if(sdk. ChangeDeviceSubnet (MAC, subnet))
{
    MessageBox.Show("Success");
}
else
{
    MessageBox.Show("Failed");
}
```

• Ingressus SDK Appendix

Ingressus SDK Parameter

System Options

Attribute Name	Parameter	Remarks
Communication Password	ComPwd	Default: null character string. Maximum: 15-bit characters (including digits and letters).
IP Address	IPAddress	Default: 192.168.1.201
Gateway	GATEIPAddress	Default value is IPAddress
BaudRate	RS232BaudRate	Default: 38400
Subnet mask	NetMask	Default: 255.255.255.0
Interlock (Door 1 and Door 2 each other is interlock. When the Door 1 in the opening , the Door 2 can only be turned off. Instead the Door 2 is opened, the Door 1 cannot be opened.) Anti-passback rule	InterLock	1: Interlock Door 1 and Door 2 mutually
(Door 1 and Door 2 each other is anti-passback, when Door 2 will be opened before Door 1 has opened , and Door 1 can't open two consecutive door)	AntiPassback	One-door Ingressus 1:Enable the anti-passback function between the readers of Door 1 Two-door Ingressus 1: Enable the anti-passback function between Door 1 and Door 2 Two-door and two-way controller 1: Enable the anti-passback function between the readers of Door 1 2: Enable the anti-passback function between the readers of Door 2 3: Enable the anti-passback function between the readers of Door 1 and between the readers of Door 2 respectively 4: Enable the anti-passback function between Door 1 and Door 2

Door Options

Attribute Name	Parameter	Remarks
Multi-card opening (The Door can be opened by more than one person through verified by, set group number of multi-card to open the door , Person in the group which is more than one authentication, set most five people)	MultiCardOpen-Door	0: Disabled 1: Enabled
Opening the door through the first card	FirstCardOpenDoor	0: Disabled 1: First-card normal open
Normal-open time segment of the door	KeepOpenTim-eZone	Default: 0 (the parameter is not set)
Punch interval	Intertime	0 means no interval (unit: second)
Verify mode	VerifyType	1:Fingerprint, 4: Card, 6:Card or fingerprint 10:Card and fingerprint 11: Card and password
Active time segment of the door (time segment in which a valid punch)	ValidTZ	Default: 0 (the door is not activated)
Duress Password	ForcePassWord	Max: 8 digits
Emergency Password	SupperPassWord	Max: 8 digits
Lock at door closing	CloseAndLock	1: Enabled, 0: Disabled
Door sensor type	SensorType	0: Not available, 1: Normal open, 2: Normal closed
Lock driver time length	Drivertime	The value range is 0 to 255. 0: Normal closed, 255: Normal open 1 to 254: Door-opening duration
Timeout alarm duration of door magnet	Detectortime	The value range is 0 to 255. Unit: second

Event Type and Code

Code	Event Types	Description
0	Normal Punch Open	In [Card Only] verification mode, the person has open door permission punch the card and triggers this normal event of open the door.
1	Punch during Normal Open Time Zone	At the normally open period (set to normally open period of a single door or the door open period after the first card normally open), or through the remote normal open operation, the person has open door permission punch the effective card at the opened door to trigger this normal events.
2	First Card Normal Open (Punch Card)	In [Card Only] verification mode, the person has first card normally open permission, punch card at the setting first card normally open period but the door is not opened, and trigger the normal event.
3	Multi-Card Open (Punching Card)	In [Card Only] verification mode, multi-card combination can be used to open the door. After the last piece of card verified, the system trigger this normal event.
4	Emergency Password Open	The password (also known as the super password) set for the current door can be used for door open. It will trigger this normal event after the emergency password verified.
5	Open during Normal Open Time Zone	If the current door is set a normally open period, the door will open automatically after the setting start time, and trigger this normal event.
6	Linkage Event Triggered	When the linkage setting the system takes effect, trigger this normal event.
7	Cancel Alarm	When the user cancel the alarm of the corresponding door, and the operation is success, trigger this normal event.
8	Remote Opening	When the user opens a door from remote and the operation is successful, it will trigger this normal event.
9	Remote Closing	When the user close a door from remote and the operation is successful, it will trigger this normal event.
10	Disable Intraday Normal Open Time Zone	When the door is in Normally Open (NO) state, swipe your valid card five times through the reader or call ControlDevice to disable the NO period on that day. In this case, trigger this normal event.
11	Enable Intraday Normal Open Time Zone	When the door's NO period is disabled, swipe your valid card (held by the same user) five times through the reader or call ControlDevice to enable the NO period on that day. In this case, trigger this normal event.

Code	Event Types	Description
12	Open Auxiliary Output	If the output point address is set to a specific auxiliary output point and the action type is set enabled in a linkage setting record, then this normal event will be triggered as long as this linkage setting takes effect.
13	Close Auxiliary Output	Events that are triggered when you disable the auxiliary input through linkage operations or by calling ControlDevice.
14	Press Fingerprint Open	Normal events that are triggered after any person authorized to open the door presses his fingerprint and passes the verification in “Fingerprint only” or “Card/Fingerprint” verification modes.
15	Multi-Card Open (Press Fingerprint)	Multi-card open(Fingerprint required): normal events that are triggered when the last person opens the door with his finger-print in “Finger print” verification mode.
16	Press Fingerprint during Normal Open Time Zone	Normal events that are triggered after any person authorized to open the door presses his valid fingerprint during the NO duration (including the NO durations set for single doors and the first-card NO duration) and through remote operations.
17	Card plus Fingerprint Open	Normal events that are triggered after any person authorized to open the door swipes his card and presses his fingerprint to pass the verification in the “Card + Fingerprint” verification mode.
18	First Card Normal Open (Press Fingerprint)	Normal events that are triggered after any person authorized to open the door becomes the first one to press his fingerprint and pass the verification during the preset first-card NO duration and in either the “Fingerprint only” or the “Card/Fingerprint” verifica-tion mode.
19	First Card Normal Open (Card plus Fingerprint)	Normal events that are triggered after any person authorized to open the door becomes the first one to swipe his card and press his fingerprint to pass the verification during the preset first-card NO duration and in the “Card + Fingerprint” verification mode.
20	Too Short Punch Interval	When the interval between two card punching is less than the interval preset for the door, trigger this abnormal event.
21	Door Inactive Time Zone (Punch Card)	In [Card Only] verification mode, the user has the door open per-mission, punch card but not at the door effective period of time, and trigger this abnormal event.
22	Illegal Time Zone	The user with the permission of opening the current door, punches the card during the invalid time zone, and triggers this abnormal event.
23	Access Denied	The registered card without the access permission of the current door, punch to open the door, triggers this abnormal event.

Code	Event Types	Description
24	Anti-Passback	When the anti-pass back setting of the system takes effect, triggers this abnormal event.
25	Interlock	When the interlocking rules of the system take effect, trigger this abnormal event.
26	Multi-Card Authentication (Punching Card)	Use multi-card combination to open the door, the card verification before the last one (whether verified or not), trigger this normal event
27	Unregistered Card	Refers to the current card is not registered in the system, trigger this abnormal event.
28	Opening Timeout:	The door sensor detect that it is expired the delay time after opened, if not close the door, trigger this abnormal event
29	Card Expired	The person with the door access permission, punch card to open the door after the effective time of the access control, can not be verified and will trigger this abnormal event.
30	Password Error	Use card plus password, duress password or emergency password to open the door, trigger this event if the password is wrong.
31	Too Short Fingerprint Pressing Interval	When the interval between two consecutive fingerprints is less than the interval preset for the door, trigger this abnormal event.
32	Multi-Card Authentication (Press Fingerprint)	In either the "Fingerprint only" or the "Card/Fingerprint" verification mode, when any person presses his fingerprint to open the door through the multi-card access mode and before the last verification, trigger this event regardless of whether the verification attempt succeeds.
33	Fingerprint Expired	When any person fails to pass the verification with his fingerprint at the end of the access control duration preset by himself, trigger this event.
34	Unregistered Fingerprint	Events that are triggered when any fingerprints are not registered in the system or registered but not synchronized to the device.
35	Door Inactive Time Zone (Press Fingerprint)	Abnormal events that are triggered when any person authorized to open the door presses his fingerprint during the preset valid duration.
36	Door Inactive Time Zone (Exit Button)	Abnormal events that are triggered when any person fails to open the door by pressing the Unlock button during the preset valid duration.

Code	Event Types	Description
37	Failed to Close during Normal Open Time Zone	Abnormal events that are triggered when any person fails to close the door in NO state by calling RemoteClose.
101	Duress Password Open	Use the duress password of current door verified and triggered alarm event.
102	Opened Accidentally	Except all the normal events (normal events such as user with door open permission to punch card and open the door, password open door, open the door at normally open period, remote door open, the linkage triggered door open), the door sensor detect the door is opened, that is the door is unexpectedly opened.
103	Duress Fingerprint Open	Use the duress fingerprint of current door verified and triggered alarm event.
200	Door Opened Correctly	When the door sensor detects that the door has been properly opened, triggering this normal event.
201	Door Closed Correctly	When the door sensor detects that the door has been properly closed, triggering this normal event.
202	Exit button Open	User press the exit button to open the door within the door valid time zone, and trigger this normal event.
203	Multi-Card Open (Card plus Fingerprint)	Normal events that are triggered when any person passes the verification with his card and fingerprint in multi-card access mode.
204	Normal Open Time Zone Over	After the setting normal open time zone, the door will close automatically. The normal open time zone include the normal open time zone in door setting and the selected normal open time zone in first card setting.
205	Remote Normal Opening	Normal events that are triggered when the door is set to the NO state for remote opening operations.
206	Device Start	When the device is being activated, this normal event is triggered.
220	Auxiliary Input Disconnected	When any auxiliary input point breaks down, this normal event is triggered.
221	Auxiliary Input Shorted	When any auxiliary input point has short circuited, this normal event is triggered.
255	Actually that obtain door status and alarm status	See Attachment 4

Ingressus SDK Error Code

Error Code	Description
-1	The command is not sent successfully
-2	The command has no response
-3	The buffer is not enough
-4	The decompression fails
-5	The length of the read data is not correct
-6	The length of the decompressed data is not consistent with the expected length
-7	The command is repeated
-8	The connection is not authorized
-9	Data error: The CRC result is failure
-10	Data error: IngressusSDK cannot resolve the data
-11	Data parameter error
-12	The command is not executed correctly
-13	Command error: This command is not available
-14	The communication password is not correct
-15	Fail to write the file
-16	Fail to read the file
-17	The file does not exist
-99	Unknown error
-100	The table structure does not exist
-101	In the table structure, the Condition field does not exit
-102	The total number of fields is not consistent
-103	The sequence of fields is not consistent
-104	Real-time event data error
-105	Data errors occur during data resolution.
-106	Data overflow: The delivered data is more than 4 MB in length
-107	Fail to get the table structure
-108	Invalid options
-201	LoadLibrary failure
-202	Fail to invoke the interface
-203	Communication initialization fails
-206	Start of a serial interface agent program fails and the cause generally relies in inexistence or occupation of the serial interface.

Error Code	Description
-301	Requested TCP/IP version error
-302	Incorrect version number
-303	Fail to get the protocol type
-304	Invalid SOCKET
-305	SOCKET error
-306	HOST error
-307	Connection attempt failed
10035	Resources temporarily unavailable. This error is returned from operations on nonblocking sockets that cannot be completed immediately, for example, recv (Wsapiref_2i9e.asp) when no data is queued to be read from the socket. It is a non-fatal error, and the operation should be retried later. It is normal for WSAEWOULDBLOCK to be reported as the result from calling connects on a nonblocking SOCK_STREAM socket (Wsapiref_8m7m.asp), since some time must elapse for the connection to be established.
10038	An operation was attempted on something that is not a socket. Ether the socket handle parameter did not reference a valid socket, or for select, a member of an fd_set was no valid.
10054	Connection reset by peer. An existing connection was forcibly closed by the remote host. This normally results if the peer application on the remote host is suddenly stopped, the host is rebooted, the host or remote network interface is disabled, or the remote host uses a hard close (See setsockopt (Wsapiref_94aa.asp) for more information on the SO_LINGER option on the remote socket). This error may also result if a connection was broken due to keep-alive activity detecting a failure while one or more operations are in progress. Operations that were in progress fail with WSAENETRESET. Subsequent operations fail with WSAECONNRESET. Connection timed out.
10060	A connection attempt failed because the connected party did not properly respond after a period of time, or established connection failed because connected host has failed to respond.
10061	Connection refused. No connection could be made because the target machine actively refused it. This usually results from trying to connect to a server that is inactive on the foreign host — that is, one with no server application running.
10065	No route to host. A socket operation was attempted to an unreachable host. See WSAENETUNREACH.