SDK FOR SPACECODE DEVICE ON TCP/IP

PROGRAMMING GUIDE

AUTHOR : CHRISTOPHE RAOULT

VERSION 1.4

CREATION DATE : JUNE 2013

Table 1: Document Change History

Document	Author	Date
Preliminary document v.1.0	C. Raoult	2012-06-05
Update new Feature	C. Raoult	2014-03-27
Update new Feature	C. Raoult	2014-10-28
Update Light Function	C. Raoult	2015-09-07

CONTENT

1. INTRODUCTION

1.1. Purpose

The SPACECODE SDK Programming guide explains client application requirements and ways of use to operate the SPACECODE Devices over Ethernet. The SDK was written for Developers who create Client Applications to help them of the understandings of the basics concept.

1.1. Scope

This library implements a Dynamically Linked Library (DLL) in .Net for Windows. The framework must be the .NET 2.0 SP1 or above.

The name of the high level main library is *TcpIP_class.dll.*

This document describes the optimal way to structure an application that uses the SPACECODE devices.

The main concern of this library is to be able to create a TCP Client able to communicate and manage device.

To be able to add user, the document assume that the code of the badge card is known.
Contact Spacecode if not the case.

2. INSTALLATION

The SDK is given with a sample project in C# and a BIN directory where the library is located.

Each library from the BIN folder has to be added to the .NET project environment using the Microsoft command *Add Reference*.

The library to be added:

TcpIP_class.dll: contains all the basic data class and structure (always required)

DataClass.dll: contains all the basic data class and structure (always required)

*SDK_SC_Fingerprint.dll :*Contains the function to drive an enroll a user from a fingerprint sensor. This is not use for medical device but mandatory as a part of the DLL for devices.

The driver of the fingerprint is added in the SDK folder and have to be install.

Just click on the executable and follow the install by clicking on next.

3. SENDING AND RECEIVING DATA

This following information described what will be exchange on the network. This function are already implemented in the high level function.

The way of sending is always the same. A command is sent on via a TCP/IP client that was prior connected to an IP address and a particular port. The default port use is the 6901. This command is several argument divide by the character ";".The number of argument depend on the command requested

The frame is defined as follow:

Command;Device_Serial_Number;Data_Argument_1; Data_Argument_2;….; Data_Argument_N

As This implementation is fully coded in the .net, only no .net Developer need the information below:

In order to not lose any data, before each command, the number byte is sent in first to inform the server the amount of data to read. The same process is use for receiving data from the device. The amount of data is sent in a fixed format of 4 bytes.

Example of implementation.

```
  private void SendData(TcpClient tcpclnt, string Data)
  {
     Stream stm = tcpclnt.GetStream();
     ASCIIEncoding asen = new ASCIIEncoding();
     byte[] data = asen.GetBytes(Data);
     int len = data.Length;
     byte[] prefix = BitConverter.GetBytes(len);
     stm.Write(prefix, 0, prefix.Length); // fixed 4 bytes
     stm.Write(data, 0, data.Length);

  }

private int GetData(TcpClient tcpclnt, out string Data)
{
  Data = null;
  Stream stm = tcpclnt.GetStream();
  byte[] readMsgLen = new byte[4];
  stm.Read(readMsgLen, 0, 4);

  int dataLen = BitConverter.ToInt32(readMsgLen, 0);
  byte[] readMsgData = new byte[dataLen];

  int dataRead = 0;
  do
  {
          dataRead += stm.Read(readMsgData, dataRead, dataLen - dataRead);
  } while (dataRead < dataLen);

  Data = System.Text.Encoding.ASCII.GetString(readMsgData, 0, dataLen);
  return 1;
}
```

4. PROPERTY AND RETURN VALUE FROM METHODS

4.1. Return value from methods

Each method use return a RetCode to inform of the status of the execution of the command requested.
This Retcode can be one of the following enumerations:

```
public enum RetCode
    {
       RC_UnknownError = -2,
       RC_FailedToConnect = -1,
       RC_Succeed = 1,
       RC_Failed = 0,
    }
```

If the function succeeds the RetCode is *RC_Succeed.*
If the function involved data, they will be in the parameters of the function.

If the function fails, the RetCode is *RC_Failed.*
The property *RetCodeStr* return a string that describe the error.

If the error is unknown, the RetCode will be *RC_UnknownError.*
This error occurs if an exception occurs on the server.

If the method cannot communicate with the device, the function returns *RC_FailedToConnect*.
In this case check the IP address, and the port, check if the device is running or if a firewall doesn't block the communication.

4.2. Received data

```
public string ReceivedData {get }
```

The Property ReceivedData contains the last data received from the server. This data contain the error definition when a method returns *RC_Failed* RetCode that can be get after received the error.

5. BASICS METHODS

The methods define below have a .net part for user that use the client .net and a low level part to let other developer the possibility to create their own client.

Some request to the server are reserve for .net user as it use serialized .net object. In case of this command is .net reserved, the same command name finish by "STR" is reserved for other user.

5.1. Restartdevice

.Net Client Method

Definition:

```
RetCode restartDevice(string strIP, int port)
```

This method allows restarting the application in the device. In case of failure of communication with the device, this method will kill the existing process and restart a fresh new one in order to renew the connection to the device.

Parameters:

string strIP:IP Address of the device

int port : Port of the device (default 6901)

Return:

A retCode as define in 4.1

5.2. Reboot

.Net Client Method

Definition:

```
RetCode rebootDevice(string strIP, int port)
```

This method allows rebooting the device. In case of failure of communication with the device and the restart application doesn't solve the issue; this method will kill the existing process and reboot the device in order to renew the connection to the device.

Parameters:

string strIP:IP Address of the device

int port : Port of the device (default 6901)

Return:

A retCode as define in 4.1

5.3. Pingserver

.Net Client Method

Definition:

RetCode pingServer(string strIP, int port)

This method allows testing the communication with the server. If succeed, the server is reachable. This function is the first method to use before try to manage a device in order to assure that the communication is present. This method informs only on the server status and in any case that the RFID device is ready.

Parameters :

string strIP:IP Address of the device

int port : Port of the device (default 6901)

Return :

A retCode as define in 4.1

5.4. Pingdevice

.Net Client Method

Definition:

```
RetCode pingDevice(string strIP, int port, string serialRFID)
```

This method allows assuming that a particular device is present and ready to work. A succeed return to this function assume that the device is connected to the server and can be managed.

Parameters:

string strIP:IP Address of the device

int port : Port of the device (default 6901)

string serialRFID : string of the serial number of the device. This unique number is the base address of a device. (8 characters string)

Return:

A retCode as define in 4.1

5.5. Getdevice

.Net Client Method

Definition:

```
RetCode getDevice(string strIP, int port, out rfidPluggedInfo[] pluggedDevice)
```

This method allows retrieving the serial number of rfid device on a particular server. The method return an array of a struct named rfidPluggedInfo that contain the following variables.

```
public class rfidPluggedInfo
   {
      public DeviceType deviceType;
      public string SerialRFID;
      public string portCom;
   }
```

Parameters:

string strIP:IP Address of the device

int port : Port of the device (default 6901)

string serialRFID : string of the serial number of the device. This unique number is the base address of a device. (8 characters string)

Return:

A retCode as define in 4.1

out rfidPluggedInfo[] pluggedDevice : An array of the device discovered.

5.6. Getstatus

.Net Client Method

Definition:

RetCode getStatus(string strIP, int port, string serialRFID, out string status)

This method allows retrieving the status of the device.

Parameters:

string strIP: IP Address of the device

int port : Port of the device (default 6901)

string serialRFID : string of the serial number of the device. This unique number is the base address of a device. (8 characters string)

Return:

A retCode as define in 4.1

out string status : a string that contain the status of the device which is one of the following values:

"DS_NotReady":	Reader not ready, reader attempt to connect to server.
"DS_Ready":	Reader Ready to work.
"DS_DoorOpen":	Reader has a door open if applicable (not SBR)
"DS_InScan":	Reader is in scan.
"DS_WaitTag":	Reader in the wait tag mode (only for SBR).
"DS_InError":	Reader is in error.

5.7. Getstatuswithnumberoftag

.Net Client Method

Definition:

public RetCode getStatusWithNumberOfTag(string strIP, int port, string serialRFID, out string status, out int nbTag)

This method allows retrieving the status of the device and retrieves the number of tag actually read on the device. Use this method to retrieve the status of the device after a scan request to detect the end of a scan and during a scan to recover the variable nbTag to have a dynamic counter of the tag present on the device during the scan process.

Parameters:

string strIP:IP Address of the device

int port : Port of the device (default 6901)

string serialRFID : string of the serial number of the device. This unique number is the base address of a device. (8 characters string)

Return:

A retCode as define in 4.1

out string status : a string that contain the status of the device which is one of the following values:

"DS_NotReady": Reader not ready, reader attempt to connect to server
"DS_Ready": Reader Ready to work
"DS_DoorOpen": Reader has a door open
"DS_InScan": Reader is in scan
"DS_WaitTag": Reader in the wait tag mode (only for SBR)
"DS_InError": Reader is in error

out int nbTag : Integer of the number of tag detecting for the last scan started.

5.8. Requestscan

.Net Client Method

Definition:

RetCode requestScan(string strIP, int port, string serialRFID)

This method allows starting a scan process of the device. The scan progress is launch but developer has to be care of the status by the *getStatusWithNumberOfTag* method (see 5.6) method to control the device status and the *getLastScan* method (see 5.15) to retrieve the result of the status.
A best case is to after request a scan, periodically request a getStatusWithNumberOfTag to control if the device is always in scan and retrieve the number of tag already scanned.
When the status of the device returns to ready, perform a *getLastScan* (see 5.15) to retrieve the list of the tag scanned.

Parameters:

string strIP: IP Address of the device

int port : Port of the device (default 6901)

string serialRFID : string of the serial number of the device. This unique number is the base address of a device. (8 characters string)

Return:

A retCode as define in 4.1

5.9. Requestscanandwait

.Net Client Method

Definition:

RetCode requestScanAndWait(string strIP, int port, string serialRFID, out InventoryData ScanResult)

This method allows starting a scan process of the device. The scan progress is launch and the function is blocking until the scan process end. If the scan is well finish the tag are available in the variable ScanResult. It is a class of the following format.

If the scan failed, the retCode will be to *RC_Failed* value.

The structure of the data is as follow:

```
public string serialNumberDevice = null;
public DateTime eventDate = DateTime.Now;
public bool bUserScan = false;
public string userFirstName = null;
public string userLastName = null;
public int nbTagAll = 0;
public int nbTagPresent = 0;
public int nbTagAdded = 0;
public int nbTagRemoved = 0;
public ArrayList listTagAll = new ArrayList();
public ArrayList listTagPresent = new ArrayList();
public ArrayList listTagAdded = new ArrayList();
public ArrayList listTagRemoved = new ArrayList();
```

- SerialNumberDevice : the serial number of the device which perform the scan
- eventDate : Date and hour of the scan
- bUserScan : if true the scan was launch with the fingerprint if false the scan comes from the scan device method.
- userFirstName : null if bUserScan to false or first name of the user.
- userLastName : null if bUserScan to false or last name of the user.
- nbTagAll : Number of tag inside the volume or area.
- nbTagPresent : number of tag that was already present from the previous scan.
- nbTagAdded : number of tag that was already added from the previous scan.
- nbTagemoved : number of tag that was already removed from the previous scan.
- listTagAll : list of tag UID of all the tag inside the active volume or area
- listTagPresent : list of tag UID that was already present from the previous scan.
- listTagAdded : list of tag UID that was already added from the previous scan.
- listTagRemoved : list of tag UID that was already removed from the previous scan.

Parameters:

string strIP: IP Address of the device

int port : Port of the device (default 6901)

string serialRFID : string of the serial number of the device. This unique number is the base address of a device. (8 characters string)

Return:

A retCode as define in 4.1

out InventoryData ScanResult: The result of the scan in case of succeed scan.

5.10. Requeststopscan

.Net Client Method

Definition:

RetCode requestStopScan(string strIP, int port, string serialRFID)

This method allows requesting the device to stop the current scan.

Parameters:

string strIP: IP Address of the device

int port : Port of the device (default 6901)

string serialRFID : string of the serial number of the device. This unique number is the base address of a device. (8 characters string)

Return:

A retCode as define in 4.1

5.11. Requestgetlastscan

.Net Client Method

Definition:

RetCode requestGetLastScan(string strIP, int port, string serialRFID, out InventoryData ScanResult)

This method allows retrieving the last successful scan performed by the device.
The result return the scan is an Inventory Class. This class is defined as follow:

```
public string serialNumberDevice = null;
public DateTime eventDate = DateTime.Now;
public bool bUserScan = false;
public string userFirstName = null;
public string userLastName = null;
public int nbTagAll = 0;
public int nbTagPresent = 0;
public int nbTagAdded = 0;
public int nbTagRemoved = 0;
public ArrayList listTagAll = new ArrayList();
public ArrayList listTagPresent = new ArrayList();
public ArrayList listTagAdded = new ArrayList();
public ArrayList listTagRemoved = new ArrayList();
```

- SerialNumberDevice : the serial number of the device which perform the scan
- eventDate : Date and hour of the scan in utc time.
- bUserScan : if true the scan was launch with the fingerprint if false the scan comes from the scan device method.
- userFirstName : null if bUserScan to false or first name of the user.
- userLastName : null if bUserScan to false or last name of the user.
- nbTagAll : Number of tag inside the volume or area.
- nbTagPresent : number of tag that was already present from the previous scan.
- nbTagAdded : number of tag that was already added from the previous scan.
- nbTagemoved : number of tag that was already removed from the previous scan.
- listTagAll : list of tag UID of all the tag inside the active volume or area
- listTagPresent : list of tag UID that was already present from the previous scan.
- listTagAdded : list of tag UID that was already added from the previous scan.
- listTagRemoved : list of tag UID that was already removed from the previous scan.

Parameters:

string strIP: IP Address of the device

int port : Port of the device (default 6901)

string serialRFID : string of the serial number of the device. This unique number is the base address of a device. (8 characters string)

Return:

A retCode as define in 4.1

out InventoryData ScanResult: The result of the scan in case of succeed scan.

5.12. Getlastdatescan

.Net Client Method

Definition:

RetCode getLastDateScan(string strIP, int port, string serialRFID, out DateTime LastDateScan)

This method allows retrieving the UTC date of the last successful scan performed by the device. in a TimeDate format.

Parameters:

string strIP: IP Address of the device

Int port : Port of the device (default 6901)

string serialRFID : string of the serial number of the device. This unique number is the base address of a device. (8 characters string)

Return:

A retCode as define in 4.1

out DateTime LastDateScan: The date of the last successful scan

5.13. Getscanfromdate

.Net Client Method

Definition:

RetCode getScanFromDate(string strIP, int port, string serialRFID, DateTime Date, out InventoryData[] ScanResult)

This method allows retrieving an array successful scan performed by the device since the date pass in parameter in uUTC value.
The result return the scan is an Inventory Class. This class is defined as follow:

```
public string serialNumberDevice = null;
public DateTime eventDate = DateTime.Now;
public bool bUserScan = false;
public string userFirstName = null;
public string userLastName = null;
public int nbTagAll = 0;
public int nbTagPresent = 0;
public int nbTagAdded = 0;
public int nbTagRemoved = 0;
public ArrayList listTagAll = new ArrayList();
public ArrayList listTagPresent = new ArrayList();
public ArrayList listTagAdded = new ArrayList();
public ArrayList listTagRemoved = new ArrayList();
```

- SerialNumberDevice : the serial number of the device which perform the scan
- eventDate : Date and hour of the scan in utc time.
- bUserScan : if true the scan was launch with the fingerprint if false the scan comes from the scan device method.
- userFirstName : null if bUserScan to false or first name of the user.
- userLastName : null if bUserScan to false or last name of the user.
- nbTagAll : Number of tag inside the volume or area.
- nbTagPresent : number of tag that was already present from the previous scan.
- nbTagAdded : number of tag that was already added from the previous scan.
- nbTagemoved : number of tag that was already removed from the previous scan.
- listTagAll : list of tag UID of all the tag inside the active volume or area
- listTagPresent : list of tag UID that was already present from the previous scan.
- listTagAdded : list of tag UID that was already added from the previous scan.
- listTagRemoved : list of tag UID that was already removed from the previous scan.

Parameters:

string strIP: IP Address of the device

int port : Port of the device (default 6901)

string serialRFID : string of the serial number of the device. This unique number is the base address of a device. (8 characters string)

Return:

A retCode as define in 4.1

out InventoryData[] ScanResult: An array of the scan from the parameter data.

5.14. REQUESTSTARTLIGHTING

Function to light a list of tag in the fridge.

.Net Client Method

Definition:

```
public RetCode RequestStartLighting(string strIP, int port, List<String> tagsToLight)
```

Parameters:

string strIP: IP Address of the device

int port : Port of the device (default 6901)

`List<String> tagsToLight` : List of string of the uid to light

Return:

A retCode as define in 4.1

When function is finished , the List<String> tagsToLight will contain all the tags that were no able to be lighted for any reason.
In a normal operation, this list is emptied saying that all tag were well lighted.
In an error operation, list of tag not lighted are kept inside this list.
This allow to inform user how many tags were lighted (difference to the initial list before run the light process) and have the uid that have issue.

5.15. REQUESTSTOPLIGHTING

Function to stop the light process.

Net Client Method

Definition:

```
public RetCode RequestStopLighting(string strIP, int port)
```

Parameters:

string strIP: IP Address of the device

int port : Port of the device (default 6901)

Return:

A retCode as define in 4.1

5.16. GETUSERLIST

The user is define per a unique first name and last name. A specific class is given with this information and the fingerprint template and/or the badge access code to create a complete user bale to be loaded and granted to a device.

.Net Client Method

Definition:

```
RetCode getUserList(string strIP, int port, string serialRFID, out UserClassTemplate[] user)
```

This method allows retrieving an array of all the user information stored in the database of the device.
This method return an array of UserClassTemplate that contains all the necessary information To a particular user define as follow :

```
public class UserClassTemplate
    {
        public string firstName;
        public string lastName;
        public string template;
        public bool[] isFingerEnrolled = new bool[10];
        public string BadgeReaderID;
    }
```

An user is defined by its firstname and lastname, the template is the binary stream from an to give to the enrolment function in the fingerprint enrolment library.

Parameters:

string strIP: IP Address of the device

int port : Port of the device (default 6901)

string serialRFID : string of the serial number of the device. This unique number is the base address of a device. (8 characters string)

Return:

A retCode as define in 4.1

out `UserClassTemplate user`: An array of the user information.

5.17. ADDUSERFROMTEMPLATE

.Net Client Method

Definition:

```
RetCode addUserFromTemplate(string strIP, int port, string
FirstName, string LastName, string template)
```

This method allows creating or modifying an user with the new enrolment or updated fingerprint template from the enrolment function. This is the base method to create a user even if the template of fingerprint is null.

Parameters:

string strIP: IP Address of the device

int port : Port of the device (default 6901)

string FirstName : The firstname of the user
string LastName : The last name of the user
string template : TheTemplate to add or to update.

Return:

A retCode as define in 4.1

5.18. ADDUSERGRANT

.Net Client Method

Definition:

```
RetCode addUserBadge(string strIP, int port, string FirstName,
string LastName, string BadgeReaderID)
```

This method allows creating or modifying the badge property of a particular user.

Parameters:

string strIP: IP Address of the device

int port : Port of the device (default 6901)

string FirstName : The firstname of the user
string LastName : The last name of the user
string BadgeReaderID : the unique badge reader code to add or to update.

Return:

A retCode as define in 4.1

5.19. DELETEUSER

.Net Client Method

Definition:

```
deleteUser(string strIP, int port, string FirstName, string
LastName, string serialRFID)
```

This method allows deleting an existing user in the device database.

Parameters:

string strIP: IP Address of the device

int port : Port of the device (default 6901)

string FirstName : The firstname of the user
string LastName : The last name of the user
string template : TheTemplate to add or to update.

Return:

A retCode as define in 4.1

5.20. ADDUSERGRANT

.Net Client Method

Definition:

```
RetCode addUserGrant(string strIP, int port, string FirstName,
string LastName, string serialRFID)
```

This method allows granting a particular user to a device.
The user can exist in the db, if he s not granting, he will be not allowed to open the door.
The user must be granted for each device needed

Parameters:

string strIP: IP Address of the device

int port : Port of the device (default 6901)

string FirstName : The first name of the user
string LastName : The last name of the user
string serialRFID : the serial number of the device that have to be granted

Return:

A retCode as define in 4.1

5.21. DELETEUSERGRANT

.Net Client Method

Definition:

```
deleteUserGrant(string strIP, int port, string FirstName, string
LastName, string serialRFID)
```

This methods allows removing a grant for a particular user to the device.
This will allows to stop the user to open the door without removing it from the device.

Parameters:

string strIP: IP Address of the device

int port : Port of the device (default 6901)

string FirstName : The first name of the user
string LastName : The last name of the user
string serialRFID : the serial number of the device

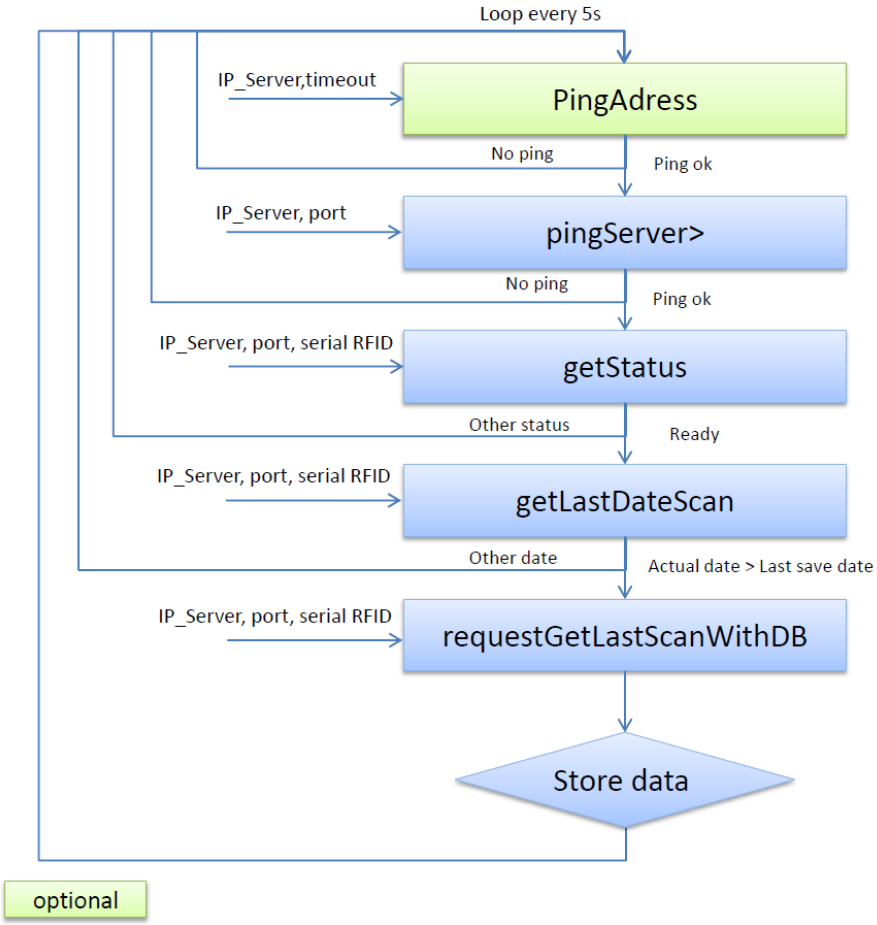
Return:

A retCode as define in 4.1

6. Basic way of getting Inventory

POLLING METHOD

The basics and simpler way to get the inventory is to create a polling loop to the device in order to get the last scan event. The basic time is a loop of 5 seconds implementing the following algorithm.



Ping RFID device to ensure a CPU is connected

Ping RFID device to ensure device operation are ok

Recover status of the device

Date of the last scan in db

Recover last full inventory data

```
if (TcpIP_class.tcpUtils.PingAddress(IP_Server, 1000))
        {
            TcpIP_class.TcpIpClient tcp = new TcpIP_class.TcpIpClient();
            if (tcp.pingServer(IP_Server, Port_Server) == TcpIP_class.TcpIpClient.RetCode.RC_Succeed)
            {
                // Device can be marked as connected here
                string status;
                if (tcp.getStatus(IP_Server, Port_Server, SerialRFID, out  status) == TcpIP_class.TcpIpClient.RetCode.RC_Succeed)
                {
                    if (status.Equals("READER_NOT_EXIST"))
                    {
                        MessageBox.Show("Bad IP for device " + SerialRFID + "\r\n Change remote configuration device");
                        continue;
                    }
                    DeviceStatus = netDeviceStatus = (DeviceStatus)Enum.Parse(typeof(DeviceStatus), status, true);
                    if (netDeviceStatus == DeviceStatus.DS_Ready)
                    {
                        DateTime lastScanIP;
                        if (tcp.getLastDateScan(IP_Server, Port_Server, SerialRFID, out lastScanIP) == TcpIP_class.TcpIpClient.RetCode.RC_Succeed)
                        {
                            if (lastScanIP > networkDeviceArray[i].lastProcessInventoryGmtDate)
                            {
                                InventoryData invData = null;
                                TcpIP_class.TcpIpClient.RetCode ret = tcp.requestGetLastScanWithDB(IP_Server, Port_Server, SerialRFID, out invData);
                                if (ret == TcpIP_class.TcpIpClient.RetCode.RC_Succeed)
                                {
                                    if (invData != null)
                                    {
                                        DateTime dtUtc = DateTime.SpecifyKind(invData.eventDate, DateTimeKind.Utc);
                                        invData.eventDate = dtUtc.ToLocalTime();
                                        // Store data here
                                    }
                                }
                            }
                        }
                    }
                }
            }
            else
            {
                // Device not connected
            }
        }
```

NOTIFICATION METHOD

The TcpIpClient Class embed a notification server that allow to create an event handler on a specific IP Address and Port.

User can choose to develop his own notification server or use the embedded one in the class.

To setup the device server information to send notification the following command have to be used.

Get the Tcp Information from the device

```
public RetCode GetTcpServerNotificationInfo(string strIP, int port, out bool bEnable, out
string hostIp, out int hostPort)

string strIP : Ip Of the device
int port : Port of the device
out bool bEnable : return if notification is enable or disable
out string hostIp : Ip of the host where the notification are sent
out int hostPort : port of the notification server where the notifications are sent
```

Return:

A retCode as define in 4.1

Get the Tcp server Information to the device

```
public RetCode SetTcpServerNotificationInfo(string strIP, int port, bool bEnable, string
tcpServerIp, int tcpServerPort)


string strIP : Ip Of the device
int port : Port of the device
bool bEnable : Enable or disable the notification
out string hostIp : Ip of the host where the notification are sent
out int hostPort : port of the notification server where the notifications are sent
```

Return:

A retCode as define in 4.1

Enable or disable the notification process.

```
public RetCode SetTcpServerNotificationOnOff(string strIP, int port, bool bEnable)

string strIP : Ip Of the device
int port : Port of the device
bool bEnable : Enable or disable the notification
```

Return:

A retCode as define in 4.1

Request the device to send a test notification

```
public RetCode TestTcpServerNotification(string strIP, int port, out bool bTestResult, out
string ExceptionMessageError)


string strIP : Ip Of the device
int port : Port of the device
out bool bTestResult : Boolean if the notification sending succeed
out string ExceptionMessageError : string of the error message if the sending failed.


To use the notification server process, a server has to be created.
```

For that the TcpIp Class has a

```
     // Instantiate a notifications handler.

_notificationServer = new TcpNotificationServer(_portNumber + 1);
_notificationServer.TcpNotifyEvent += new
TcpNotificationServer.TcpNotifyHandlerDelegate(HandleNotificationEvent);
_notificationServer.StartServer();
```

This will create a notification server on the port number and the notification sent from the device will be received in the handleNotification function.

The handle notification receive a `rfidTcpNotArg` argument

```
public class rfidTcpNotArg : EventArgs
    {
        public rfidTcpNotArg(string serialNumber, rfidTcpNotArg.ReaderTcpNotify RNValue);
        public rfidTcpNotArg(string serialNumber, rfidTcpNotArg.ReaderTcpNotify RNValue, int
scanId);
        public rfidTcpNotArg(string serialNumber, rfidTcpNotArg.ReaderTcpNotify RNValue,
double lastTempBottle, double lastTempChamber);

        public double LastTempBottle { get; }
        public double LastTempChamber { get; }
        public rfidTcpNotArg.ReaderTcpNotify RN_Value { get; }
        public int ScanId { get; }
        public string SerialNumber { get; }

        public enum ReaderTcpNotify
        {
            RN_ScanStarted = 3,
            RN_ScanCompleted = 4,
            RN_ScanCancelledByHost = 5,
            RN_Door_Opened = 64,
            RN_Door_Closed = 65,
            RN_TempEvent = 67,
            RN_TempEventChanged = 68,
            RN_TestNotification = 69,
        }
    }
```

A simple way to handle the notification is given under.

```
private void HandleNotificationEvent(Object sender, rfidTcpNotArg arg)
```

```
        {
            switch (arg.RN_Value)
            {
                case rfidTcpNotArg.ReaderTcpNotify.RN_ScanStarted:
                    break;

                case rfidTcpNotArg.ReaderTcpNotify.RN_ScanCompleted:
                    break;

                case rfidTcpNotArg.ReaderTcpNotify.RN_Door_Opened:
                    break;

                case rfidTcpNotArg.ReaderTcpNotify.RN_ScanCancelledByHost:
                    break;
                case rfidTcpNotArg.ReaderTcpNotify.RN_TempEventChanged:
                case rfidTcpNotArg.ReaderTcpNotify.RN_TempEvent:
                    break;
                case rfidTcpNotArg.ReaderTcpNotify.RN_TestNotification:
                    break;

            }
        }
```

NOTIFICATIONS

`rfidTcpNotArg.ReaderTcpNotify.RN_ScanStarted`

Occurs when a new scan start.
Device send on the TCP server the following frame

```
"CR_DISPATCH CC_SB_SCAN_STARTED " + _myLocalIp + " " + _myPort + " " + _serialRfid;

_myLocalIp : Ip Of the device that generate the notification
_myPort : port o the device
_serialRfid: Serial Number of the device
```

`rfidTcpNotArg.ReaderTcpNotify.RN_ScanCompleted`

Occurs when a new scan start is completed
Device send on the TCP server the following frame.

```
"CR_DISPATCH CC_SB_NEWINV " + _myLocalIp + " " + _myPort + " " + _serialRfid + " " +
_idScanEvent;

_myLocalIp : Ip Of the device that generate the notification
_myPort : port o the device
_serialRfid: Serial Number of the device
_idScanEvent : Unique scan number ID
```

While receiving this notification the receiver can request the last scan with the ID by

```
  InventoryData lastInventory = new InventoryData();
  TcpIpClient.RetCode response = _tcpClient.requestGetScanFromIdEvent(_ipAddress,
_portNumber, _serialNumber, scanId, out lastInventory);
```

`TcpNotificationType.ScanCancelByHost:`

Occurs when a new scan start is cancelled by a host

Device send on the TCP server the following frame.

```
"CR_DISPATCH CC_SB_SCAN_CANCEL_BY_HOST " + _myLocalIp + " " + _myPort + " " + _serialRfid;

_myLocalIp : Ip Of the device that generate the notification
_myPort : port o the device
_serialRfid: Serial Number of the device
```

```
TcpNotificationType.TestTcp:
Occurs when a test notification is requested
```

Device send on the TCP server the following frame.

```
CR_DISPATCH CC_SB_TEST_TCP " + _myLocalIp + " " + _myPort + " " + _serialRfid;
```

```
TcpNotificationType.TempChanged
Occurs when the temperature change in the fridge (temperature is updated every minute to
avoid loading the network)
```

Device send on the TCP server the following frame.

```
"CR_DISPATCH CC_SB_TEMP_CHANGED " + _myLocalIp + " " + _myPort + " " + _serialRfid + " " +
_temp;

_myLocalIp : Ip Of the device that generate the notification
_myPort : port o the device
_serialRfid: Serial Number of the device
_temp : Double of the current temperature
```

7. Creating and grant a user

Create and grant an user for a device cabinet required to use 3 different functions.
The first thing is to create the user with the AddUserFromTemplate function.
If the user is using a badge, the function AddUserBadge need to be used.
At the end , to grant user, function AddGrantUser have to be used.

These three methods and all the delete one allow to have a full control of the user in the device.
A user can be created or deleted from the device.
The grant function allows to grant or not a particular user for a period time, the user still exists in the device but not allowed to open the door.

Example of implementation

```
if ((!string.IsNullOrEmpty(textBoxFirstName.Text)) &&
(!string.IsNullOrEmpty(textBoxLastName.Text)))
            {
                userInEnroll = new DeviceGrant();
                userInEnroll.user.firstName = textBoxFirstName.Text;
                userInEnroll.user.lastName = textBoxLastName.Text;
                userInEnroll.user.BadgeReaderID = textBoxReaderCard.Text;
                userInEnroll.userGrant = (UserGrant)comboBoxGrant.SelectedIndex;
                //userInEnroll.user.template = myFinger.EnrollUser(null,
userInEnroll.user.firstName, userInEnroll.user.lastName, null, false);

                tcp = new TcpIP_class.TcpIpClient();
```

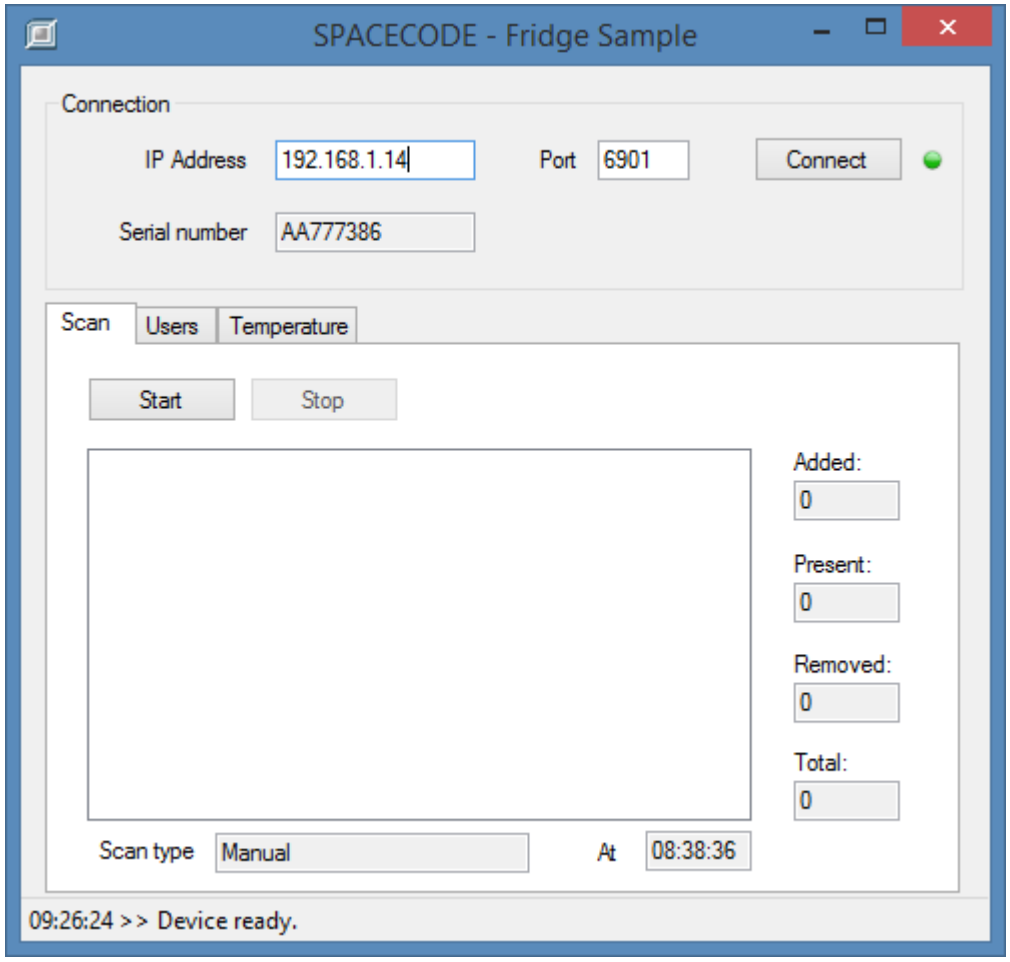
```
                TcpIpClient.RetCode ret = tcp.addUserFromTemplate(ip, port,
userInEnroll.user.firstName,
                                                  userInEnroll.user.lastName,
userInEnroll.user.template);

                if (!string.IsNullOrEmpty(userInEnroll.user.BadgeReaderID))
                {
                    tcp.addUserBadge(ip, port, userInEnroll.user.firstName,
userInEnroll.user.lastName, userInEnroll.user.BadgeReaderID);
                }


                if (comboBoxGrant.SelectedIndex != 0)
                    ret = tcp.addUserGrant(ip, port, userInEnroll.user.firstName,
userInEnroll.user.lastName, serialRFID, userInEnroll.userGrant);
                else
                    ret = tcp.deleteUserGrant(ip, port, userInEnroll.user.firstName,
userInEnroll.user.lastName, serialRFID);

                updateListBoxUser();
            }
            else
                MessageBox.Show("Please Enter FirstName and LastName Field before Enroll",
"Info", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
```

8. DEMO APPLICATION

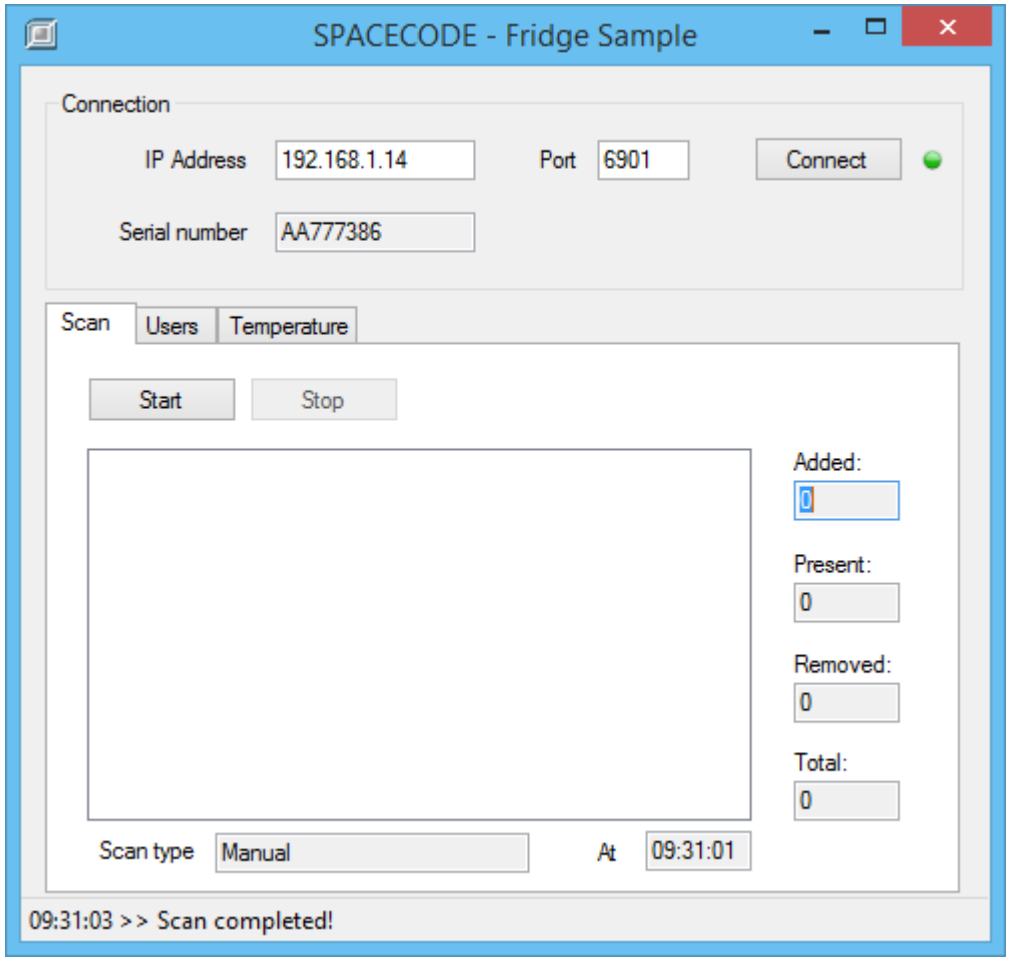
8.1. Application

A small sample application is provided to show an example of the code implementation.
Please contact SPACECODE to get last version.

To use this sample application, fill in first the Ip address and the port (default 6901) of the device you want to manage.
Click on Connect to connect to the device , get this status and get the last inventory . The status will be displayed in the status bar.

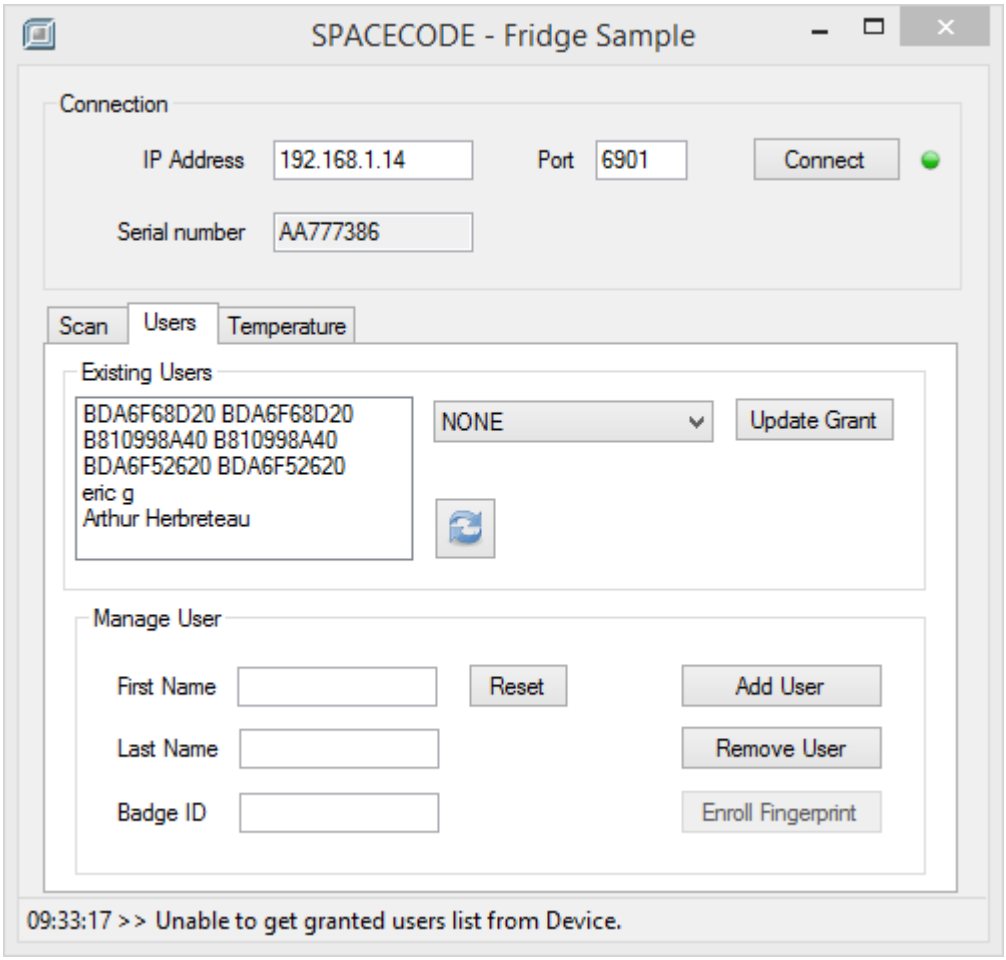


Click On scan Device to perform a scan.
The scan will request the status of the reader periodically and will display at the end the last scan result

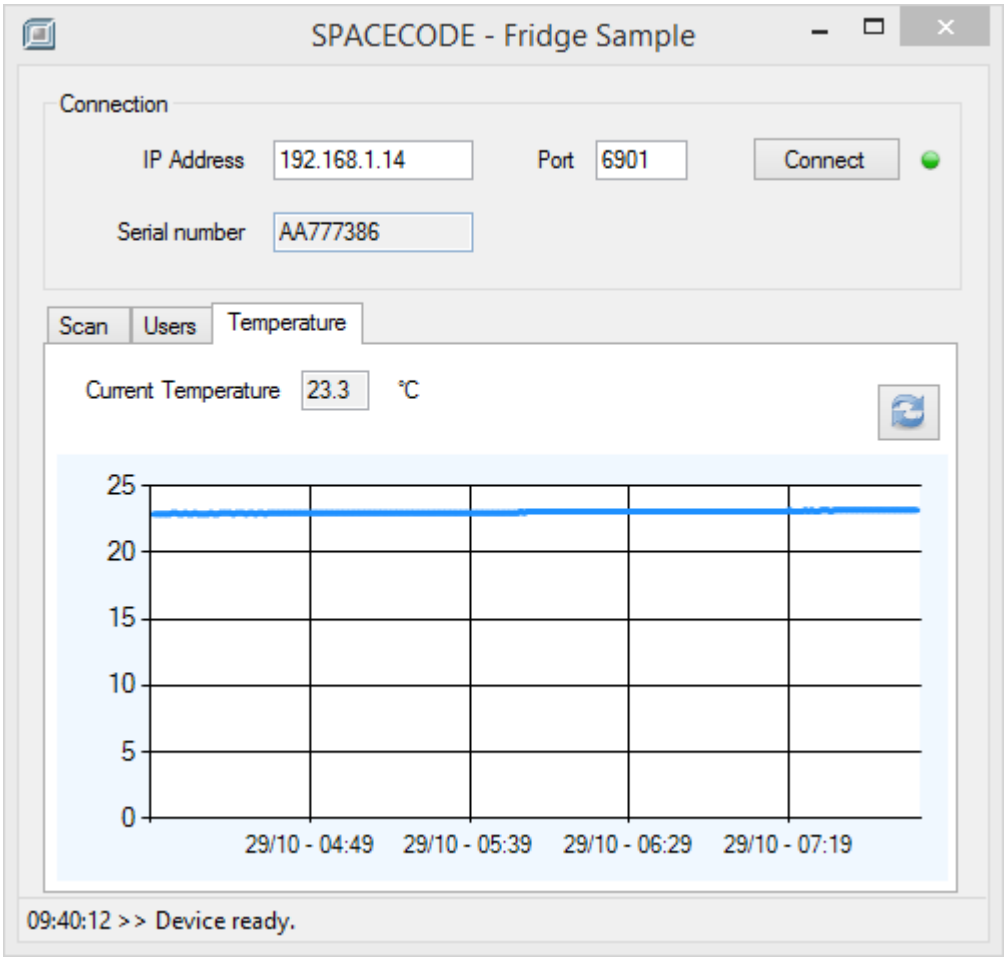


The scan info are updated.

On the User tab, All the function are present to manage user with a access card and/or a fingerprint.
To be used the fingerprint driver has to be installed.



A last tab show the current temperature and a graph display last temperature for the last 4 hours. The current hour is not displayed has device store the temperature for a complete hour.


SPACECODE - Fridge Sample
Connection
IP Address
192.168.1.14
Port
6901
Connect
Serial number
AA777386
Scan
Users
Temperature
Current Temperature
23.3
°C
25
20
15
10
5
0
29/10 - 04:49
29/10 - 05:39
29/10 - 06:29
29/10 - 07:19
09:40:12 >> Device ready.