Add-on PCL for iOS

Migration and Backward Compatibility

ICO-OPE-02153 V2
Restricted

Ingenico
2014/10/21

www.ingenico.com
28/32, boulevard de Grenelle, 75015 Paris - France / (T) +33 (0)1 58 01 80 00 / (F) +33 (0)1 58 01 91 35
Ingenico – S.A. au capital de 53 086 309 € / 317 218 758 RCS PARIS

Contents

Ingenico document - Reproduction or disclosure prohibited without written authorization
Please check document validity before using

1 Introduction

PCL for iOS 2.0 introduces a major change in the way the network communication is managed between the iOS device and the Telium device.

This change has been made in order to expose the following feature to the Telium device:

- Ability to use multiple simultaneous connections through the iOS device. One of the main application is the ability to use the FTP protocol between Telium and any other device connected to the network
- Ability to use a native IP network with the connected iOS device, working in both direction, in order to be fully compliant with architecture and application developed for Android or Windows devices

In order to implement these features, a major change has been done in PCL for iOS architecture.
To avoid putting current project at risk, the backward compatibility with previous versions (on both PCL side and SDK side) has been kept, **but any new project must use the new architecture**, as defined in the “PCL for iOS integration guide”

The goal of this document is to provide necessary information for backward compatibility.

Thanks to this new architecture, some components no longer needed are now deprecated. They will be definitively removed from Ingenico delivery end of 2015 and support will be stopped.
These components are:

- - ICNetwork channel from the iSMP framework
- - Lib IPBox

An intermediary deprecated period will be set up, to help you identify what must no longer be used.

Ingenico document - Reproduction or disclosure prohibited without written authorization
Please check document validity before using

2 Managing backward compatibility

2_1 Overview

The table below describes the various ways the network communication can be managed, depending on the SDK version and the PCL add-on version.

	Telium SDK < 9.18	**Telium SDK >= 9.18**
PCL Add-On < 2.00	Legacy case. Use the ICNetwork channel to manage all communications	Legacy use case only: ICNetwork to manage all communications
PCL Add-On >= 2.00	Legacy use case only: ICNetwork to manage all communications	Communication can be managed in 2 ways: • Legacy, with the use of the ICNetwork channel • Native IP, with the use of the ICPPP channel

2_2 Choosing the mode

In the 1.* version of the PCL Add-on, the TCP/IP connectivity is emulated using a dedicated channel iOS ICNetwork.
To reduce the compatibility issues, the behavior of this channel has not been changed and the real TCP/IP implementation will work alongside. However, it will not be possible to use them both (either at the same time and one after the other).
The choice of the TCP/IP communication will be determine by the use (or not) of the ICNetwork channel. When opened by the application the Telium device will use the legacy mode.

To help project migration, the function “**getSPMCIVersion**” is available on iOS to determine whether the Telium SDK supports the new network architecture.
If the string returned is “0252” or higher, then the new architecture can be used. Else, the legacy channel ICNetwork must be used.

2_3 Working in Legacy mode

You will be able to keep working in legacy mode with no change at the application level (on both Telium and iOS side) with the new version of the SDK and the PCL Add-on (although note that since the iOS libraries are statically linked, if you want to use the new PCL Add-on, you will have to rebuild the application).

Ingenico document - Reproduction or disclosure prohibited without written authorization
Please check document validity before using

Detailed information on how to work in legacy mode can be found in the section 3.

2_4 Using the native TCP/IP connectivity

Refer to PCL for iOS integration for details about using the native TCP/IP connectivity.
To summarize, if you want to use the new communication features provided by this add-on, you will be able to use one of the following 3 ways on Telium:

- BSD socket, which will be become natively supported
- LinkLayer, by using the configuration LL_PHYSICAL_V_PCL. This configuration can be for all supported mobile device (tablet / smartphone, Android / iOS / Windows)
- CB2A dll (France region), as it will be updated to use the Link Layer configuration.

In addition, the iOS application will be able to access a TCP server located on Telium side.
In order to access the TCP server on Telium side, you will need to configure a bridge between iOS and Telium that will tell PCL which port must be redirected to Telium. The bridge configuration will be done using a function provided in PCL library.

Ingenico document - Reproduction or disclosure prohibited without written authorization
Please check document validity before using

3 Legacy mode: ICNetwork channel (deprecated)

THIS METHOD IS NO LONGER TO BE USED AND WILL NO LONGER BE AVAILABLE AND SUPPORTED END OF 2015

3_1 How it works

The “**ICNetwork**” class provides network access to the Companion by using the available network interfaces of the iOS device (Wi-Fi or 3G). This class does not have any method that the applications can interact with. It does only provide a number of callbacks that allow the application to keep track of the ongoing connections. The protocol within which these callbacks are defined is named “**ICNetworkDelegate**”.Please refers to doxygen documentation for more information about class initialization.

This class works behind the scenes as a proxy. It receives the following requests from the Companion on the serial link or Bluetooth connection in case of iSMP-C and iCMP:

- Connect(socketID, host, port)
- Write(socketID)
- Close(socketID)

“**ICNetwork**” works like a proxy. It creates real network sockets on the iOS side, opens a connection to the targeted host and functions as an intermediary between the remote host and the Companion.

The Companion can only open one connection at a given time.
When the iOS device and the Companion lose synchronization, all the open connections and sockets are closed.

3_1_1 Requirement

“**ICNetwork**” requires “**ICAdministration**” to work. Indeed, the latter provides, behind the scenes, the available iOS device’s network interfaces to the Companion when asked for that. Therefore, it’s necessary to initialize “**ICAdministration**” too, when initializing “**ICNetwork**”.

3_1_2 ICNetwork Disposal

To dispose the network communication channel, the iOS application has just to release the “**ICNetwork**” object that it has created.

3_1_3 Remote host connection state

The setServerConnectionState function of the class **ICAdministration** can be used to personalize the Telium header by changing the connectivity icon ▛. Note that when using this function, the whole synchronization between the header and the connection status must be managed by your application.
The table below shows the icon that will appear depending on state of the connection that is specified.

State	Icon

Ingenico document - Reproduction or disclosure prohibited without written authorization
Please check document validity before using

Connected to Server	■
No Connected to Server	▪

3_1_4 ICNetworkDelegate Protocol

This protocol provides a set of callbacks to be implemented by the delegate of “**ICNetwork**”. These are listed in the table below and will be detailed one by one just after.

Callback	**Calling Context**
(void)networkWillConnectToHost:(NSString*)host onPort:(NSUInteger)port	Random Runloop
(void)networkDidConnectToHost:(NSString*)host onPort:(NSUInteger)port	Random Runloop
(void)networkFailedToConnectToHost:(NSString*)host onPort:(NSUInteger)port	Random Runloop
(void)networkDidDisconnectFromHost:(NSString*)host onPort:(NSUInteger)port	Random Runloop
(void)networkDidReceiveErrorWithHost:(NSString*)host andPort:(NSUInteger)port	Random Runloop
(void)networkData:(NSData*)data incoming:(BOOL)isIncoming	Random Runloop

For more information about all the callbacks please refer to the doxygen documentation.

3_2 **Checking the connection state between the device and the Companion**

The “**ICISMPDevice”** class has properties and methods that provide information about a communication channel and help keep track of the state of the connection between the iOS device and the Companion.

Among those, the following class method checks if the Companion and the iOS device are connected: ***(BOOL)isAvailable***

If the iOS device is linked with the Companion and the authentication between the two devices went well (the connection state icon on the Companion screen header is full:■), this method return **YES**, otherwise it returns **NO.**

The other properties are inherited by the derived classes, mentioned previously, and should be called on instances of those classes.

Name	**Type**	**Description**
(BOOL)isAvailable	Property	Indicated if a channel is open and ready
(NSString *)protocolName	Property	Returns the protocol name of a certain channel
(NSInputStream *)inStream	Property	Returns a pointer to the input stream used to read data from the Companion on a given channel
(NSOutputStream *)outStream	Property	Returns a pointer to the output stream used to write data to the Companion on a given channel

Ingenico document - Reproduction or disclosure prohibited without written authorization
Please check document validity before using

The most important property among those is ***isAvailable***. This property must be tested by the application after initializing a communication session. The returned value determines whether the session is open and that the Companion is ready to exchange data and reply to requests.

The ***inStream*** and ***outStream*** properties can be useful to debug communication issues and get the state of the serial streams opened to the Companion, although it is very unlikely to experience such issues. Those properties are however essential for communication on the **Transaction** channel that does not, like the four others, provide a high level API to write commands to the Companion, but provides instead raw read/write streams that can be used to send customized commands or to implement a customized communication protocol between the two devices. This part will be detailed in “**ICTransaction**” class.

Connection State Changes

It is possible and recommended to keep track of connection state changes after opening a communication session. Indeed, the connection between the two devices may be lost unpredictably. This may happen at different situations:

- The two devices are unlinked (Bluetooth connection broken or iOS device detached in case of iSMP or iCMP),
- the Companion reboots to apply and update,
- The iOS device goes into sleep mode and closes the communication channels.

During such scenarios, the iOS application and the iSMP library objects that handle the communication remain alive. They should then be able to detect the next time that the Companion is available to re-establish the communication if necessary. This can be done by implementing the two following methods defined within the “**ICISMPDeviceDelegate**” protocol:

Callback	**Calling Context**
(void)accessoryDidConnect:(ICISMPDevice *)sender	**Main Runloop**
(void)accessoryDidDisconnect:(ICISMPDevice *)sender	**Main Runloop**

Those two methods are called on the delegate of an instance of “**ICISMPDevice**” to notify it with Companion’s connection and disconnection events. The “**sender**” argument is a pointer to the source of the event, which is a pointer to an “**ICISMPDevice**” object. The “**sender**” parameter is important in case the delegate implements different protocols at the same time (for example “**ICPrinterDelegate**” and “**ICNetworkDelegate**”). In this case, it becomes necessary to be able to distinguish which channel has connected or disconnected.

The two callbacks are both called on the main run-loop of the program. This means that it is safe to manipulate User Interface objects where they are implemented.

accessoryDidConnect callback may be called at different times depending on the class that triggered it. This may happen either before or after a communication channel is opened:

- The callback is called to signal the presence of the Companion, which means that the application can opens the communication session. Examples of classes that have this behavior are “**ICBarCodeReader**” and “**ICAdministration**”.
- The callback is called to signal that the communication channel associated with the **sender** object provided as argument has been opened and ready. “**ICPrinter**”, “**ICSPP**”, “**ICNetwork**” and “**ICTransaction**” classes have this behavior. These classes open the communication session at initialization and each time the external accessory (Companion) is detected, and close it when they are released.

Ingenico document - Reproduction or disclosure prohibited without written authorization
Please check document validity before using

This Document is Copyright © 2014 by INGENICO Group. INGENICO retains full copyright ownership, rights and protection in all material contained in this document. The recipient can receive this document on the condition that he will keep the document confidential and will not use its contents in any form or by any means, except as agreed beforehand, without the prior written permission of INGENICO. Moreover, nobody is authorized to place this document at the disposal of any third party without the prior written permission of INGENICO. If such permission is granted, it will be subject to the condition that the recipient ensures that any other recipient of this document, or information contained therein, is held responsible to INGENICO for the confidentiality of that information.

Care has been taken to ensure that the content of this document is as accurate as possible. INGENICO however declines any responsibility for inaccurate, incomplete or outdated information. The contents of this document may change from time to time without prior notice, and do not create, specify, modify or replace any new or prior contractual obligations agreed upon in writing between INGENICO and the user.

INGENICO is not responsible for any use of this software, which would be non-consistent with the present document.

All trademarks used in this document remain the property of their rightful owners.”