Technology
Solutions (UK) Ltd

www.tsl.uk.com

UHF ASCII 2.0 MOBILE SDK (iOS) USER GUIDE

www.tsl.uk.com
Design • Development • Manufacture

CONTENT

Overview

This document provides an introduction to the TSL ASCII 2.0 protocol and the .iOS API provided to simplify development of applications using the protocol.

History

Version	Date	Modifications
1.0	14/10/2013	Initial release for iOS SDK v1.0.0
1.1	09/09/2014	Updated for ASCII 2.2 support.

INTRODUCTION

ASCII 2.0 is a recreation of the ASCII protocol Technology Solutions originally created for their range of UHF readers. The aim of the original protocol was to enable rapid testing of UHF commands from a terminal command prompt connected to a Technology Solutions UHF reader. As more readers implemented the protocol the ASCII protocol became useful to support multiple platform types without having to support the full binary API.

The main objectives of the ASCII 2.0 protocol were to maintain the ease of use for experimenting at the command prompt but to add more structure to the commands and responses to make it easier to command from an application. This has been achieved with the following:

- A defined start sequence to every command (e.g. ".iv")
- A consistent way to pass parameters to a command
- Simple framing of all commands and responses with a header and line terminator
- Signalling the termination of a response with an empty line
- Signalling the error for a command with a return code "ER: xxx" or "OK"

The ASCII 2.0 protocol can be implemented in most modern languages very simply. Once a connection is established to the reader a command line is sent to the reader. Lines of response are then read from the reader until an empty line is received signalling the end of the response. The line preceding the empty line will start "OK:" or "ER: xxx" indicating whether the command executed successfully.

All ASCII commands start with a period '.' followed by two characters to identify the command e.g. ".iv" for inventory. The command is terminated with an end of line (Cr, Lf or CrLf). The rest of the command line can contain parameters with or without values. A parameter starts with a minus '-'symbol followed by one or more characters to identify the parameter then followed by the parameter value.

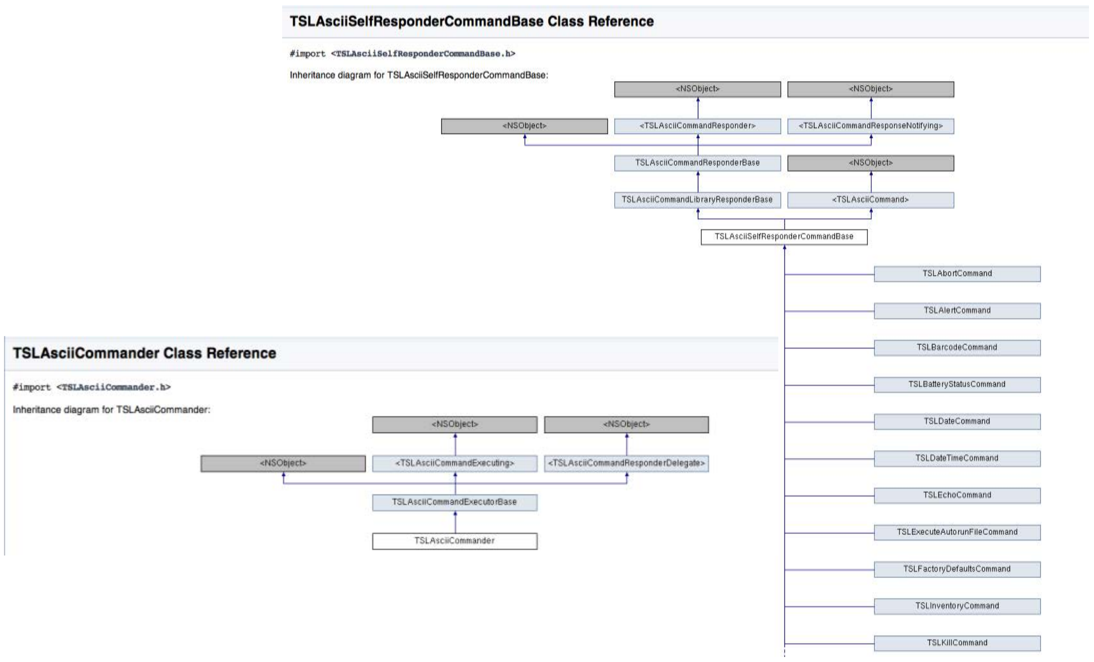
".iv –x " perform an inventory, reset the command parameters to default

An ASCII response is a sequence of lines terminated by an empty line. The each line has a two character header followed by a colon ':' the remainder of the line is the value corresponding to the header.

"ME: this is a message" is a message response as part of an ASCII response.

An ASCII response always starts with the command started "CS:" line and ends with either "OK:" if the command completed successfully or "ER: xxx" if it failed, xxx represents an error code. If the "ER:" is sent is may also be preceded by an "ME:" line with a human readable error.

API INTRODUCTION



TSLAsciiSelfResponderCommandBase Class Reference

```
#import <TSLAsciiSelfResponderCommandBase.h>
```

Inheritance diagram for TSLAsciiSelfResponderCommandBase:

<NSObject>
<NSObject>
<NSObject>
<TSLAsciiCommandResponder>
<TSLAsciiCommandResponseNotifying>
TSLAsciiCommandResponderBase
<NSObject>
TSLAsciiCommandLibraryResponderBase
<TSLAsciiCommand>
TSLAsciiSelfResponderCommandBase
TSLAbortCommand
TSLAlertCommand
TSLBarcodeCommand
TSLBatteryStatusCommand
TSLDateCommand
TSLDateTimeCommand
TSLEchoCommand
TSLExecuteAutorunFileCommand
TSLFactoryDefaultsCommand
TSLInventoryCommand
TSLKillCommand

TSLAsciiCommander Class Reference

```
#import <TSLAsciiCommander.h>
```

Inheritance diagram for TSLAsciiCommander:

<NSObject>
<NSObject>
<NSObject>
<TSLAsciiCommandExecuting>
<TSLAsciiCommandResponderDelegate>
TSLAsciiCommandExecutorBase
TSLAsciiCommander

The TSLAsciiCommands framework provides developers with a set of easy-to-configure commands that encapsulate the TSL ASCII 2 Protocol and a helper object to communicate with any compatible TSL Device. There is help documentation (HTML) to browse all the classes, protocols and members of the framework and a docset file is provided to support Xcode code completion and context-sensitive help.

COMMAND PARAMETERS

The *TSLAsciiCommand* protocol represents a command to be sent to the reader and there are classes that implement this protocol for all of the common commands. These command classes have properties to set the parameters to send to the reader.

All the parameters for a command are optional, where a parameter is not specified the reader uses its cached value of the parameter. API commands generally use enumerated parameters with a …_NotSpecified value indicating that the parameter is not specified. By default, all API command parameters are not specified.

Where commands have parameters they also have a *resetParameters* parameter to reset all parameters to their defaults in the reader before executing the command. Executing the command will update the reader's cached parameters with the values of all parameters on the command line. Commands also have a *readParameters* parameter to read the current value of all the commands parameters from the device.

Commands can also have a *takeNoAction* parameter. When specified the command parameters can be read or modified without performing the actual command (e.g. an inventory can be configured without actually performing the inventory, the inventory can then be performed by another inventory command without specifying any parameters.

EXECUTING A COMMAND AND RESPONSE HANDLING

Communication with the reader uses an instance of *TSLAsciiCommander* and it can be controlled and interrogated using instances derived from *TSLAsciiCommand*. Configured commands are sent to the TSL device using *TSLAsciiCommander*'s *executeCommand:* method.

TSLAsciiCommander handles incoming responses from the reader using instances derived from *TSLAsciiCommandResponder* (responders) in a 'responder chain' (See *responderChain* property).

Each line of ASCII data received from the reader is passed to the *processReceivedLine:* method of each responder in the chain in order. The responder can choose to process this data based on its content. If the responder recognises and processes a particular line it can choose to prevent down-chain responders from seeing it by returning YES from the *processReceivedLine:* method.

It can be useful to communicate with the Reader in a synchronous mode where a command is executed and the thread then waits until the response has been received from the reader. Most of the commands from the TSL ASCII command library support this by being derived from *TSLAsciiSelfResponderCommandBase*. These commands provide their own default response handling and use the incoming data to set appropriate properties e.g. *TSLBatteryStatusCommand* can be executed synchronously and then its *batteryLevel* property holds the current battery level of the Reader.

To use synchronous commands requires the addition of a special responder (the *synchronousResponder*) to *TSLAsciiCommander*'s responder chain. This allows developers to determine where in the responder chain synchronous commands are handled.

All commands in the library that support synchronous operation provide a factory method (*synchronousCommand*) to create a command pre-configured to operate synchronously.

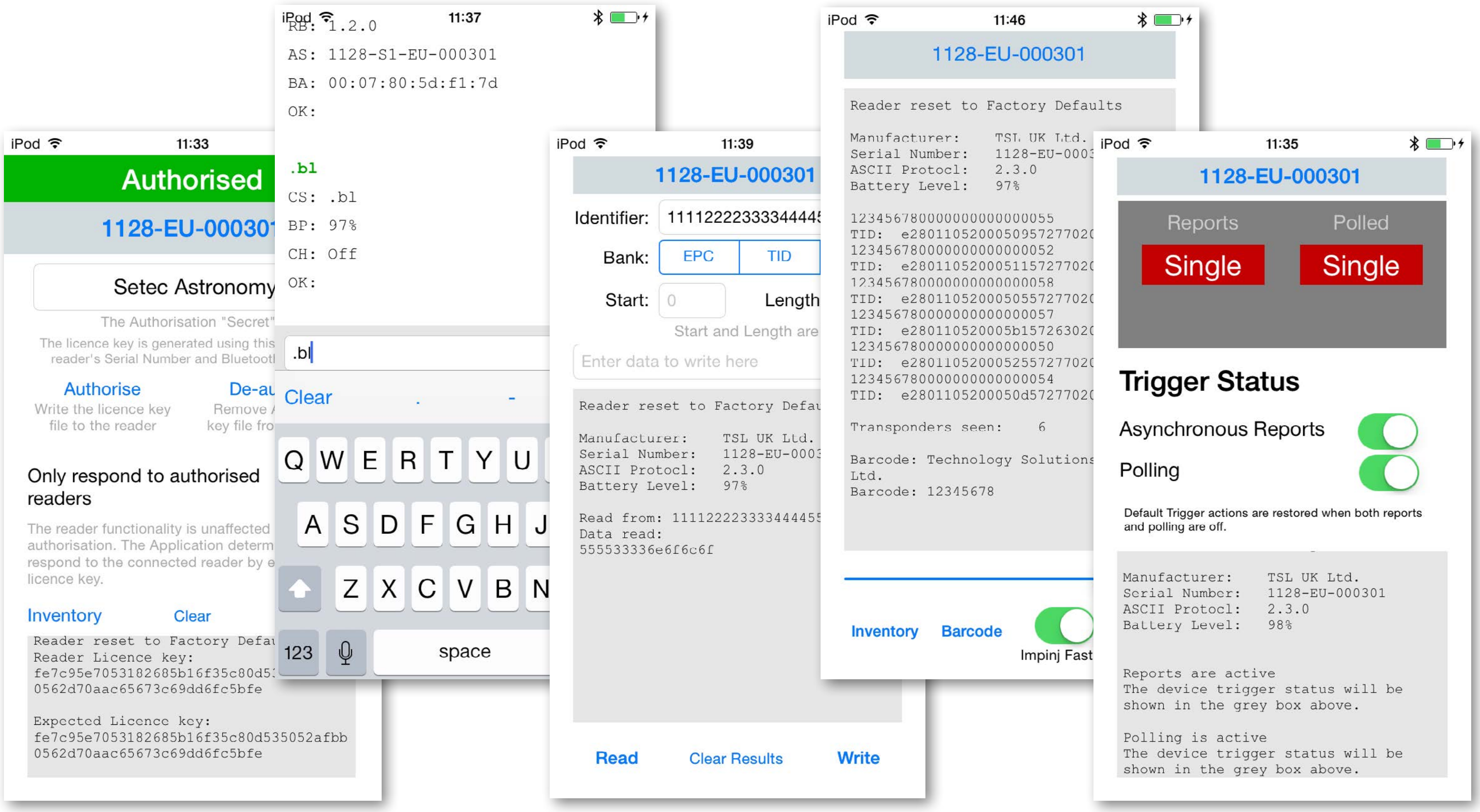
Readers can also generate incoming data through actions such as the user pulling the device trigger to initiate an inventory or barcode scan. The responder chain will see all data sent from the device no matter how it was initiated.

Handling of reader responses asynchronously can also be achieved using commands from the library. Since each command is also a responder for its own command it can be added to the responder chain to capture incoming responses e.g. a *TSLInventoryCommand* can be added to the responder chain to capture responses from any inventory commands. By default, all commands are set to only respond to commands issued through the API however, they can also be configured to accept responses initiated from any source by setting the *captureNonLibraryResponses* property to YES. This allows the capture of reader responses initiated from the device trigger switch as well as from API commands.

Commands also implement the *TSLAsciiCommandResponseNotifying* protocol which allows access to the response lifecycle events responseBegan and responseEnded. When the response line from the reader contains the CS: header, the responseBegan event is signalled. When lines starting either OK: or ER: are received the responseEnded event occurs. Blocks are used to handle these events and are configured using *responseBeganBlock* and *responseEndedBlock* properties.

Examples of all the techniques listed above can be found in the Getting Started section of the help documentation and in the SDK Sample Projects.

SAMPLE APPLICATIONS


iPod 11:33
Authorised
1128-EU-000301
Setec Astronomy
The Authorisation "Secret"
The licence key is generated using this reader's Serial Number and Bluetooth
Authorise
Write the licence key file to the reader
De-au
Remove key file fro
Only respond to authorised readers
The reader functionality is unaffected authorisation. The Application determ respond to the connected reader by e licence key.
Inventory
Clear

```
Reader reset to Factory Defau
Reader Licence key:
fe7c95e7053182685b16f35c80d53
0562d70aac65673c69dd6fc5bfe

Expected Licence key:
fe7c95e7053182685b16f35c80d535052afbb
0562d70aac65673c69dd6fc5bfe
```

```
RB: 1.2.0
AS: 1128-S1-EU-000301
BA: 00:07:80:5d:f1:7d
OK:

.bl
CS: .bl
BP: 97%
CH: Off
OK:
```

.bl
Clear
Q W E R T Y U
A S D F G H J
Z X C V B N
123
space

iPod 11:39
1128-EU-000301
Identifier: 11112222333344445
Bank: EPC TID
Start: 0
Length
Start and Length are
Enter data to write here

```
Reader reset to Factory Defau

Manufacturer:     TSL UK Ltd.
Serial Number:    1128-EU-0003
ASCII Protocl:    2.3.0
Battery Level:    97%

Read from: 11112222333344445
Data read:
555533336e6f6c6f
```

```
Reader reset to Factory Defaults

Manufacturer:     TSL UK Ltd.
Serial Number:    1128-EU-0003
ASCII Protocl:    2.3.0
Battery Level:    97%

123456780000000000000055
TID:  e2801105200050957277020
123456780000000000000052
TID:  e2801105200051157277020
123456780000000000000058
TID:  e2801105200050557277020
123456780000000000000057
TID:  e28011052000 5b157263020
123456780000000000000050
TID:  e2801105200052557277020
123456780000000000000054
TID:  e2801105200050d57277020

Transponders seen:    6

Barcode: Technology Solutions
Ltd.
Barcode: 12345678
```

```
Manufacturer:     TSL UK Ltd.
Serial Number:    1128-EU-000301
ASCII Protocl:    2.3.0
Battery Level:    98%


Reports are active
The device trigger status will be
shown in the grey box above.

Polling is active
The device trigger status will be
shown in the grey box above.
```

SAMPLE PROJECTS

The sample projects provided with the SDK demonstrate the use of the iOS API to command devices that support the TSL ASCII 2.0 protocol to perform many of the common operations required to by RFID applications including inventorying, reading and writing to transponders.

- *Inventory* quick-start sample project
  - Demonstrates the use of the *TSLInventoryCommand* & *TSLBarcodeCommand*
- *ReadWrite* quick-start sample project
  - Demonstrates the use of the *TSLReadTransponder* and *TSLWriteSingleTransponder* commands.
- *Trigger* quick-start sample project
  - Demonstrates use of the device trigger to initiate operations in the application using *TSLSwitchActionCommand*, *TSLSwitchStateCommand* and *TSLSwitchResponder*
  - Two approaches are implemented: polling the trigger state and handling asynchronous trigger state reports.
- *TSLTerm* quick-start sample project
  - This is a simple terminal program that will allow users to experiment with the raw ASCII 2.0 protocol (See the ASCII Protocol 2.0 guide for details).
- *Licence Key* quick-start sample project
  - Demonstrates addition and removal of licence keys from a reader.
  - Provides an example of storing a reader-specific cryptographic hash as the licence key.

ABOUT TSL

ABOUT

TSL designs and manufactures both standard and custom embedded, snap on and standalone peripherals for handheld computer terminals. Embedded technologies include:

- RFID - Low Frequency, High Frequency & UHF
- *Bluetooth*® wireless technology
- Contact Smartcard
- Fingerprint Biometrics
- 1D and 2D Barcode Scanning
- Magnetic Card Readers
- OCR-B and ePassport

Utilizing class leading Industrial design, TSL develops products from concept through to high volume manufacture for Blue Chip companies around the world. Using the above technologies TSL develops innovative products in a timely and cost effective manner for a broad range of handheld devices.

CONTACT

Address:	Technology Solutions (UK) Limited, Suite C, Loughborough Technology Centre, Epinal Way, Loughborough, Leicestershire, LE11 3GE. United Kingdom.
Telephone:	+44 (0)1509 238248
Fax:	+44 (0)1509 220020
Email:	enquiries@tsl.uk.com
Website:	www.tsl.uk.com

ISO 9001: 2008