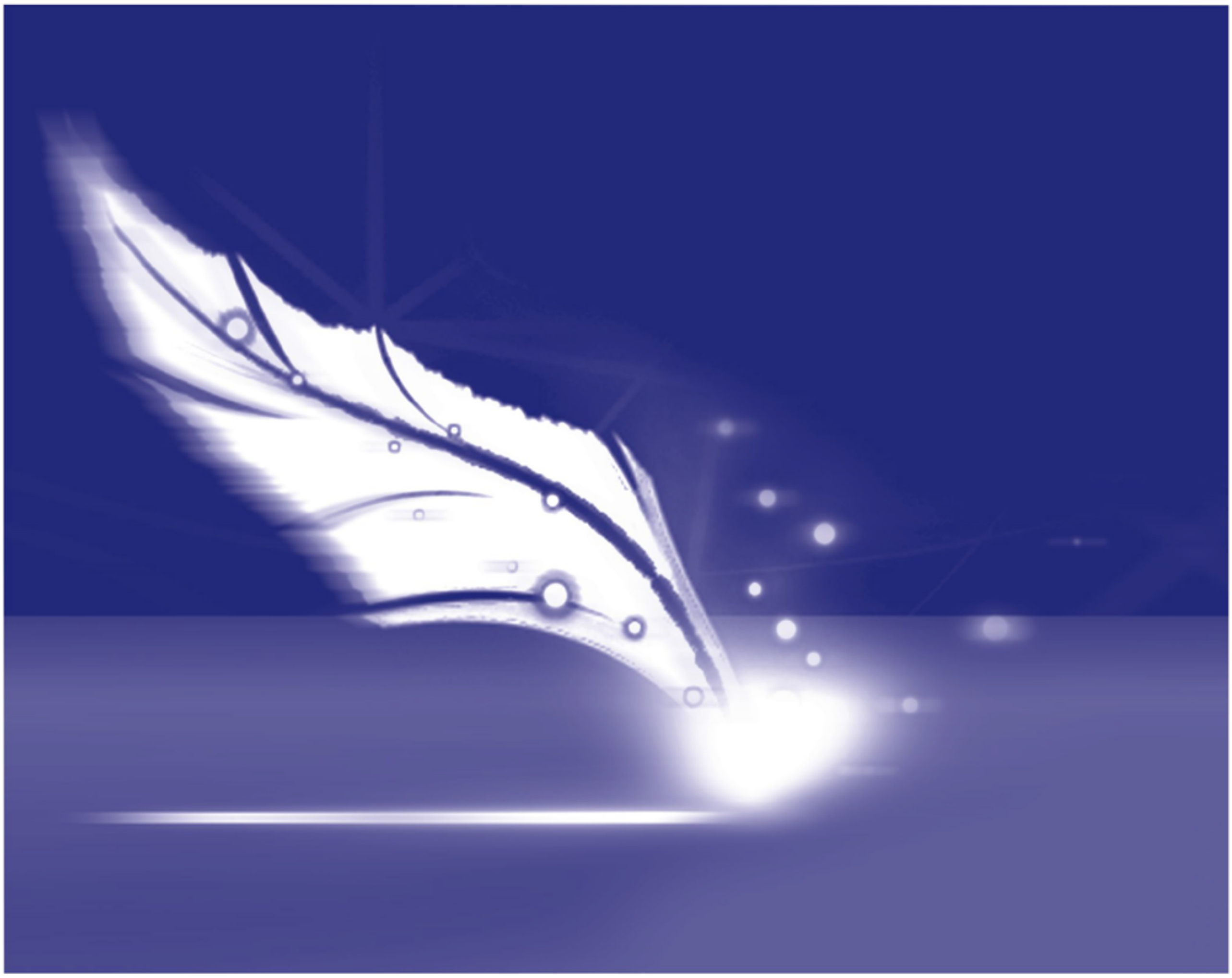
USER GUIDE



> Std RS232 Protocol

DATALOGIC

Datalogic S.r.l.
Via S. Vitalino 13
40012 – Calderara di Reno
Italy

Std RS232 Protocol User Guide

Ed.: 09/2017

Helpful links at www.datalogic.com: **Contact Us**, **Terms and Conditions**, **Support**.

Datalogic reserves the right to make modifications and improvements without prior notification.

Datalogic shall not be liable for technical or editorial errors or omissions contained herein, nor for incidental or consequential damages resulting from the use of this material.

Ed. 09/2017

REVISION INDEX

Revision	Date	Number of added or edited pages
04/2010	2010-04-13	Release
01/2012	2012-01-03	Review
02/2013	2013-02-13	Review
09/2017	2017-09-22	ii

NOTE:
We sometimes update the documentation after original publication. Therefore, you should also review the documentation at www.datalogic.com for updates.

TABLE OF CONTENTS

1. PROTOCOL

1.1 INTRODUCTION

Deployed with the Lighter Suite, you get some project examples. In order to be able to manage the laser operations, you can use the project “Std RS-232” which implements the protocol described in this document over a RS-232 link.

The protocol is compatible with the old one provided by the “std-rs232.vbs” Smartist script. The addictions are some commands (Smartist supported just three of them) and the “End of Mark signal”.

The RS-232 link grants a full duplex communication, but the protocol is designed as a master-slave, where the Lighter project represents the slave part, interpreting and executing the commands provided by a third part service.

Once started the project it opens the port (see **Table 1 - Com port initial settings**) and starts listening on it, waiting for a valid command sequence.

1.2 CUSTOMIZATION

This project is designed to provide a common application that can be used for testing laser application capabilities and as a project for easy customization, where third-party applications need to control the laser system.

The user needing this project works for its purposes should check and set:

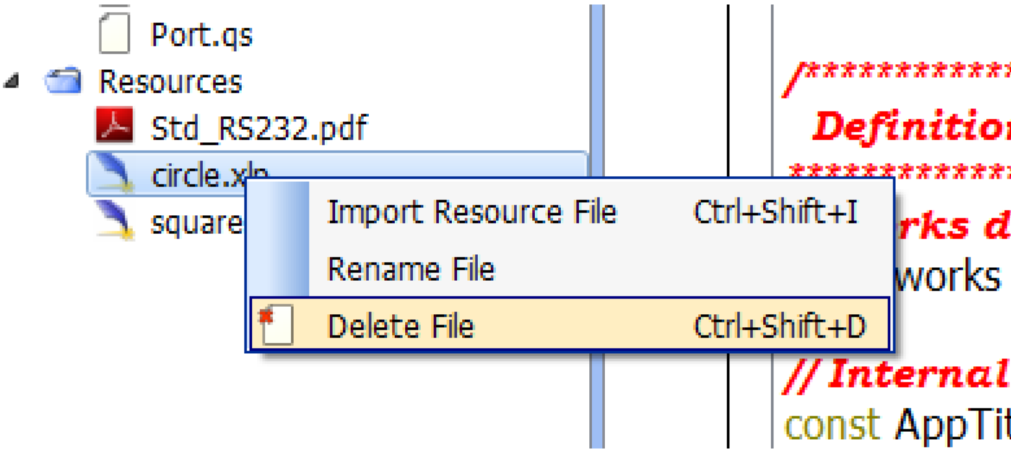
1. port settings – defined in the “function classPort” in the “Port.qs” file.
2. layouts – defined in the array **works** in the “main.qs” file.

From the project version 1.2, users can add as many layouts as they want. Please consider that layouts are all loaded when the project starts, this can result in a slow load (READY output signal can take a longer time to be set high and the graphic dialog could take more time to be showed) but once a layout selection command is received, the layout switch is done very fast.

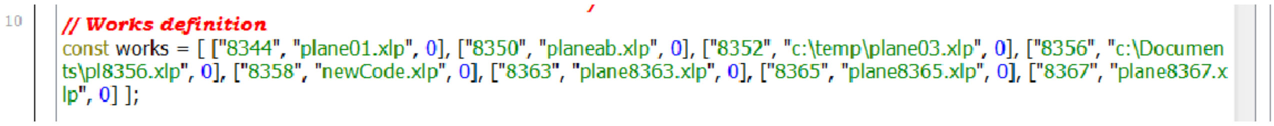
The project is deployed with two layouts as stated in Table 4 - Layouts definitions, in order to modify the project for using other prepared layouts, please follow this steps:

Prepare all needed layouts and store them as XLP files in a folder the device can access to.

Remove old layouts.

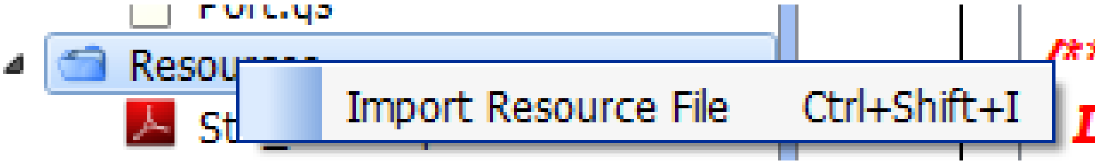


Modify the works array definitions (**main.qs** file) according to the layout you are going to use. Please take care about the three fields required for each layout, where the first one is the **ID** to be used for layout selection command (see paragraph for further details) , the second one is the file path for layout retrieval, the third one is used by the project, so leave it as zero valued.

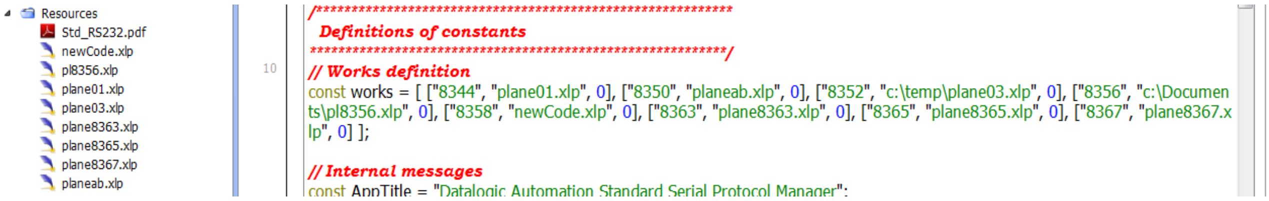


```
// Works definition
const works = [ ["8344", "plane01.xlp", 0], ["8350", "planeab.xlp", 0], ["8352", "c:\temp\plane03.xlp", 0], ["8356", "c:\Documen
ts\pl8356.xlp", 0], ["8358", "newCode.xlp", 0], ["8363", "plane8363.xlp", 0], ["8365", "plane8365.xlp", 0], ["8367", "plane8367.x
lp", 0] ];
```

In the figure above layouts there are an example with 8 layouts stored in different folders just to show the capability of the script (**plane01** has no path specification while **plane03** is indicated being in the **c:\temp** folder). You are invited storing all the planes in the layouts folder (from Editor just save document to device), or importing the planes as project resources (see picture below) and they will be collected and stored in the project folder.



Even if the previous **works** array definition is correct, you are invited using the one showed here below.



```
/****************************************************************
  Definitions of constants
****************************************************************/
// Works definition
const works = [ ["8344", "plane01.xlp", 0], ["8350", "planeab.xlp", 0], ["8352", "c:\temp\plane03.xlp", 0], ["8356", "c:\Documen
ts\pl8356.xlp", 0], ["8358", "newCode.xlp", 0], ["8363", "plane8363.xlp", 0], ["8365", "plane8365.xlp", 0], ["8367", "plane8367.x
lp", 0] ];

// Internal messages
const AppTitle = "Datalogic Automation Standard Serial Protocol Manager";
```

1.3 PROJECT DESCRIPTION

The project consists of 5 source files and 2 layout files.

- main.qs – hosts the entry point function “main”, the objects allocation and the main interaction, the layouts table.
- GUI.qs – defines the graphical user interface through the **classMainWindow** class
- Util.qs – contains some functions used in many places of the code.
- Protocol.qs – defines the start and end bytes, the protocol management and parsing methods through the **classProtocol** class.
- Port.qs – hosts the **classPort** class which allow the management of the PC COM port. It also defines the initial settings for port opening. They are:

Setting	Value
Port name	COM1
Baud rate	57600
Flow control	Off
Parity	No
Data bits	8
Stop bit	1

1 - Com port initial settings

WARNING:

Starting from project **version 2.0**, user can permanently change the COM port setting directly from the user interface without manually changing the *Port.qs* file. The COM port setting will be saved in the config.txt files present in the project folder

Using project prior to **version 2.0** is possible to change the COM port setting from the user interface but they will not be saved. User who need to change the COM port setting permanently must change manually the **classPort** function in the **port.qs** file, where the default parameters are set.

1.4 LINK LAYER

Port settings are initially set in the constructor in the Port.qs file. They can also be changed by the user with the graphic interface and applied once the port is reopened.
Each message is a byte sequence starting with the ESCAPE byte and terminating with the TERMINATOR one (see **"Table 2 - Special bytes values"**). The only exception is the byte (ENDOFMARK) sent by this client when the laser terminates the marking process.
Each byte of a message is transmitted immediately after the preceding one, with no wait between. The message is parsed once the TERMINATOR byte is received.
The client application realized with this project monitors the COM port waiting for the underlying object signalling data availability; this way the resource consumption results very low.

1.5 END OF MARKING PROCESS

When the marking process terminates, this client application sends a byte through the Com port signalling this event. The byte is set to ENDOFMARK (see **Table 2 - Special bytes values**).

1.6 THE PROTOCOL

Commands and answers of this protocol are exchanged as byte sequences each one starting with the ESCAPE byte and terminating with the TERMINATOR one.

Byte	ASCII	DEC	HEX	BIN
ESCAPE		27	1B	00011011
TERMINATOR		13	0D	00001101
SEPARATOR	,	44	2C	00101100
ENDOFMARK		7	7	00000111

2 - Special bytes values

These bytes value are defined in the Protocol.qs file in the first lines.
The second byte is the command identifier, an ASCII character indicating the command to be executed.

Command	ASCII	DEC	HEX	BIN
Layout selection	S	83	53	01010011
Text setting	D	68	44	01000100
Echo	E	69	45	01000101
Status request	T	84	54	01010100
Variable get/set	C	67	43	01000011
Start command	X	88	58	01011000
Stop command	P	80	50	01010000
Version request	V	86	56	01010110
System time request	Y	89	59	01011001
Diode time request	I	73	49	01001001

3 - Command bytes

The other part of the sequence varies according to the command specifications.

1.6.1 Layout Selection

Command:

```
ESCAPE S <layout identifier> TERMINATOR
```

This command allows the selection of a layout from those defined in the main.qs file in the array **works**.

ID	FileName
01	circle.xlp
02	square.xlp

4 - Layouts definitions

The <layout identifier> is an ASCII string reporting the ID as defined in the first column of the array **works** (see **Table 4 - Layouts definitions**).
Once the command is executed the layout is updated and showed in the preview area. No answer is sent.

Example:

Byte	Symbol	Meaning
27		ESCAPE
83	S	Layout Selection
48	0	Layout Identifier = 01
49	1	
13	CR	TERMINATOR
27		ESCAPE
68	D	Text Setting
48	0	String Identifier = 01
49	1	
44	,	SEPARATOR
72	H	String Content = Hello
101	e	
108	l	
108	l	
111	o	
13	CR	TERMINATOR

5 - Layout Selection and text setting example

1.6.2 Text setting

Command:

```
ESCAPE D <string identifier> , <string content> TERMINATOR
```

This command sets the content of the string with the ID given by the <string identifier> and the value provided by the <string content>. The string is searched in the current plane which is the first one in the **works** array (see **Table 4 - Layouts definitions**) or the one set by the.
The example is reported in **Table 5 - Layout Selection and text setting example**. No answer is sent.

1.6.3 Echo

Command:

```
ESCAPE E <byte sequence> TERMINATOR
```

Answer:

```
ESCAPE E <byte sequence> TERMINATOR
```

This command is normally used just for checking the link correct behaviour or for keep-alive function (test if the counterpart is active or not). As the client receives this kind of command, it just replies with an exact copy of the message to the sender.

In the example below is reported the command which is identical to the answer:

Byte	Symbol	Meaning
27		ESCAPE
69	E	Echo
48	0	Byte sequence
49	1	
27		
68	D	
48	0	
49	1	
44	,	
72	H	
101	e	
108	l	
108	l	
111	o	
13	CR	TERMINATOR

6 - Echo example

1.6.4 Status request

Command:

```
ESCAPE T TERMINATOR
```

Answer:

```
ESCAPE T <ASCII sequence> TERMINATOR
```

This command queries the client about the device status. Possible answer are those reported in **Table 7 - Status possible values**.

System:SystemStates	Numeric values
SYSTEM_OFF	0
SYSTEM_WARM_UP	1
SYSTEM_WAIT	2
SYSTEM_STAND_BY	3
SYSTEM_STAND_BY_SHUTTER_CLOSED	4
SYSTEM_READY	5
SYSTEM_READY_SHUTTER_CLOSED	6
SYSTEM_BUSY	7
SYSTEM_WARNING	8
SYSTEM_ERROR	9

7 - Status possible values

Example:

Byte	Symbol	Meaning
27		ESCAPE
84	T	Status Request
13	CR	TERMINATOR

8 - Status request example

Byte	Symbol	Meaning
27		ESCAPE
84	T	Status Answer
0		SYSTEM_OFF
13	CR	TERMINATOR

9 - Status answer example

1.6.5 Variable get/set

This command can get or set the content of a global variable according to the command format. If the identifier of the variable is followed by the SEPARATOR byte (see **Table 2 - Special bytes values**) and an ASCII sequence, the global variable identified by the <variable identifier> is set to a value according to the <ASCII sequence>, otherwise an answer composed as specified below is sent.
Get command:

```
ESCAPE C <variable identifier> TERMINATOR
```

Answer:

```
ESCAPE C <variable identifier> SEPARATOR <ASCII sequence> TERMINATOR
```

Byte	Symbol	Meaning
27		ESCAPE
67	C	Variable Get
88	X	Variable identifier = XX
88	X	
13	CR	TERMINATOR

10 - Variable Get Example

Byte	Symbol	Meaning
27		ESCAPE
67	C	Variable Get Answer
88	X	Variable Identifier = XX
88	X	
44	,	SEPARATOR
49	1	ASCII Sequence = 10
48	0	
13	CR	TERMINATOR

11 - Variable Get Answer Example

Set Command:

```
ESCAPE C <variable identifier> , <ASCII sequence> TERMINATOR
```

WARNING:
The master application can't set a numeric variable to an alphabetical sequence of characters and no error is returned.

Byte	Symbol	Meaning
27		ESCAPE
67	C	Variable Set
89	Y	Variable Identifier = YY
89	Y	
44	,	SEPARATOR
84	T	ASCII Sequence = Testo da seriale
101	e	
115	s	
116	t	
111	o	
32		
100	d	
97	a	
32		
115	s	
101	e	
114	r	
105	i	
97	a	
108	l	
101	e	
13	CR	TERMINATOR

12 - Variable Set Example

1.6.6 Start command

Command:

```
ESCAPE X TERMINATOR
```

This command asks the client starts marking the currently selected layout then shows it in the preview area. No answer is sent.

Example:

Byte	Symbol	Meaning
27		ESCAPE
83	S	Layout Selection
48	0	Layout Identifier = 01
49	1	
13	CR	TERMINATOR
27		ESCAPE
88	X	Start Command
13	CR	TERMINATOR
27		ESCAPE
80	P	End Command
13	CR	TERMINATOR

13 - Example sequence with Selection,Start and Stop

1.6.7 Stop command

Command:

```
ESCAPE P TERMINATOR
```

This command asks the client stops marking the layout currently in progress (if any).
The example is reported in **Table 13 - Example sequence with Selection,Start and Stop**. No answer is sent.

1.6.8 Version request

Command:

```
ESCAPE V TERMINATOR
```

Answer:

```
ESCAPE V <ASCII sequence> TERMINATOR
```

This command queries the client about the software version is running on the host. The answer carries the **Laser Engine version** the project is hosted in. The version is provided as a string of ASCII characters.

Example:

Byte	Symbol	Meaning
27		ESCAPE
86	V	Version request
13	CR	TERMINATOR

14 - Version Request Example

Byte	Symbol	Meaning
27		ESCAPE
86	V	Version Answer
53	5	ASCII Sequence = 5.2.0 alpha
46	.	
50	2	
46	.	
48	0	
32		
97	a	
108	l	
112	p	
104	h	
97	a	
13	CR	TERMINATOR

15 - Version Answer Example

1.6.9 System time request

Command:

```
ESCAPE Y TERMINATOR
```

Answer:

```
ESCAPE Y <ASCII sequence> TERMINATOR
```

This command can get the number of seconds the system has been running. This number is provided as a string of ASCII characters. It can report ‘-1’ if the device is not connected or not turned on.

Example:

Byte	Symbol	Meaning
27		ESCAPE
89	Y	System Time Request
13	CR	TERMINATOR

16 - System time request example

Byte	Symbol	Meaning
27		ESCAPE
89	Y	System Time Answer
45	-	ASCII Sequenze = -1
49	1	
13	CR	TERMINATOR

17 - System time answer example

1.6.10 Diode time request

Command:

```
ESCAPE I TERMINATOR
```

Answer:

```
ESCAPE I <ASCII sequence> TERMINATOR
```

This command can get the number of seconds the diode has been running. This number is provided as a string of ASCII characters.

Example:

Byte	Symbol	Meaning
27		ESCAPE
73	I	Diode Time Request
13	CR	TERMINATOR

18 - Diode time request example

Byte	Symbol	Meaning
27		ESCAPE
73	I	Diode Time Answer
45	-	ASCII Sequenze = -1
49	1	
13	CR	TERMINATOR

19 - Diode time answer example

DATALOGIC

www.datalogic.com