

***LaurTec***

# **RS232 Terminal**

**Debugging Embedded Systems**

**Author :** Mauro Laurenti

**ID:** PJ11004-EN

## License

**Copyright (C) 2009-2017**

**Author:** Mauro Laurenti

**Web Page:** [www.LaurTec.com](http://www.LaurTec.com)

The usage of "RS232 Terminal" software given by Mauro Laurenti (the Author) imply the acceptance of the following license:

- The Software must not be used before reading the documentation.
- The Software can be redistributed without restrictions.
- The official documentation must be distributed with the Software.
- The Software cannot be used for developing and testing military equipment.
- The licenses associated with it must be preserved.

THE SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS MATERIAL. THE AUTHOR SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

## Content

<b>Abstract</b> .....	4
<b>Applications</b> .....	4
<b>System Requirements</b> .....	5
<b>Software Installation</b> .....	6
<b>Basic Interface</b> .....	7
<b>Top Menu</b> .....	9
Menu File.....	9
Menu Edit.....	9
Menu Actions.....	10
Menu Tools.....	10
Menu Settings.....	11
Menu Help.....	11
<b>Toolbar</b> .....	12
<b>GUI Settings</b> .....	13
Start up.....	13
Colors.....	13
Font.....	14
Editor.....	14
<b>Communication Settings</b> .....	16
Port Settings.....	16
Write Settings.....	17
Read Settings.....	19
Display.....	21
<b>RS232 Signals</b> .....	23
<b>Debugging Tools</b> .....	24
ASCII Table.....	25
Data Converter.....	26
Statistics.....	26
Sliders.....	27
Macros.....	28
Check bits.....	29
<b>Bibliography</b> .....	32
<b>History</b> .....	33

## Abstract

The RS232 protocol is a quite old standard, nevertheless it is far from being obsoleted. Many embedded systems still rely on that protocol as easy and powerful tool for debugging and data exchange. For instance Linux based embedded systems often use it as backdoor terminal for debugging. The USB standard could not ignore the hegemony of its hold predecessor, hence, within the CDC class (Communication Device Class) defined in the USB standard, there is a portion of it dedicated to the RS232 emulation. Many systems, even if are complying with USB standard are actually designed to handle a virtual RS232 communication port. Data exchange with such devices can still be performed using the old tools for RS232. One of the most used program is without any doubts HyperTerminal, designed by Microsoft and included with all the Windows distributions...till Windows XP. HyperTerminal disappearance let other free programs, to take over. Even if it seams that there is no room any longer for RS232 free tools, whoever has used any of those, would have been glad having on hands more features, normally provided only by commercial Software.

Transmitting and reading data in HEX format, sending multiple word packets, Data Parsing, auto send, data formatting, are some of the features that come with *RS232 Terminal*, making it one of the most flexible tool for RS232 debugging...and all come for Free.

## Applications

*RS232 Terminal* allows testing and debugging any embedded system based on the following protocols:

- RS232
- RS485
- RS422
- USB CDC Class
- UART based communication

depending on the protocol, a specific transceiver/adaptor might be required to properly interface with the bus under test.

## System Requirements

*RS232 Terminal* has been designed using Microsoft Visual Studio Community 2015, using Visual Basic.

### Required .NET framework

To properly run the application, it is required the .NET Framework Version 3.5. If you do not have it already installed within your PC, you can download it from Microsoft web page. You can check if the right version of .NET Framework is already installed on your machine, just checking the list of the installed programs within the Configuration Panel. Another simple way to see if the right .NET Framework is already installed, it is just by running the *RS232 Terminal* application. If it does not boot you do not have the .NET Framework installed (easy and dirty test).

### Operative System

Currently the *RS232 Terminal* has been tested on the following operative systems, but it may run on others:

- Windows 7 64 bits
- Windows 10 64 bits



#### **Nota**

While the application was first developed under Windows XP, this operative system is no longer supported, since Microsoft does not support it.

### Hardware Requirements

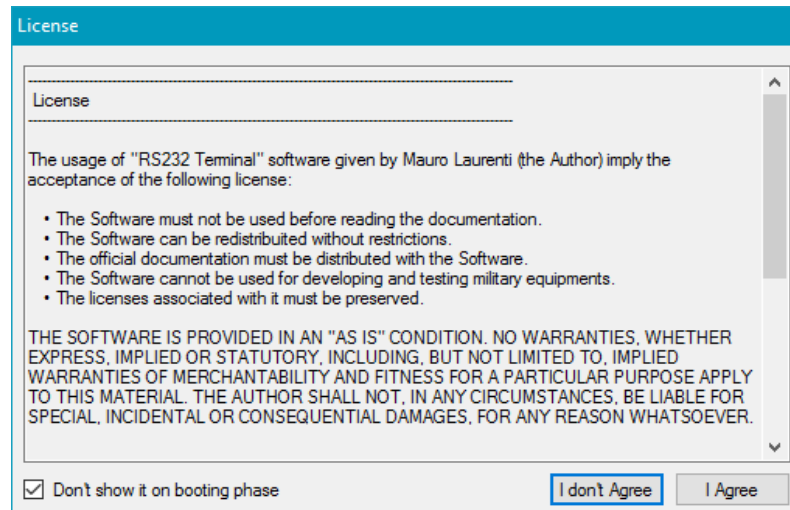
One RS232 port available, either emulated or true.

### Microprocessor

Rule of thumb, if you have resources to run Windows 7 you should be fine!

## Software Installation

No software installation is required. Just copy the application folder containing all the files, within any path you want to run the application from. The first time you will execute the program you will be prompted to accept or refuse the license agreement, as shown in Figure 1.

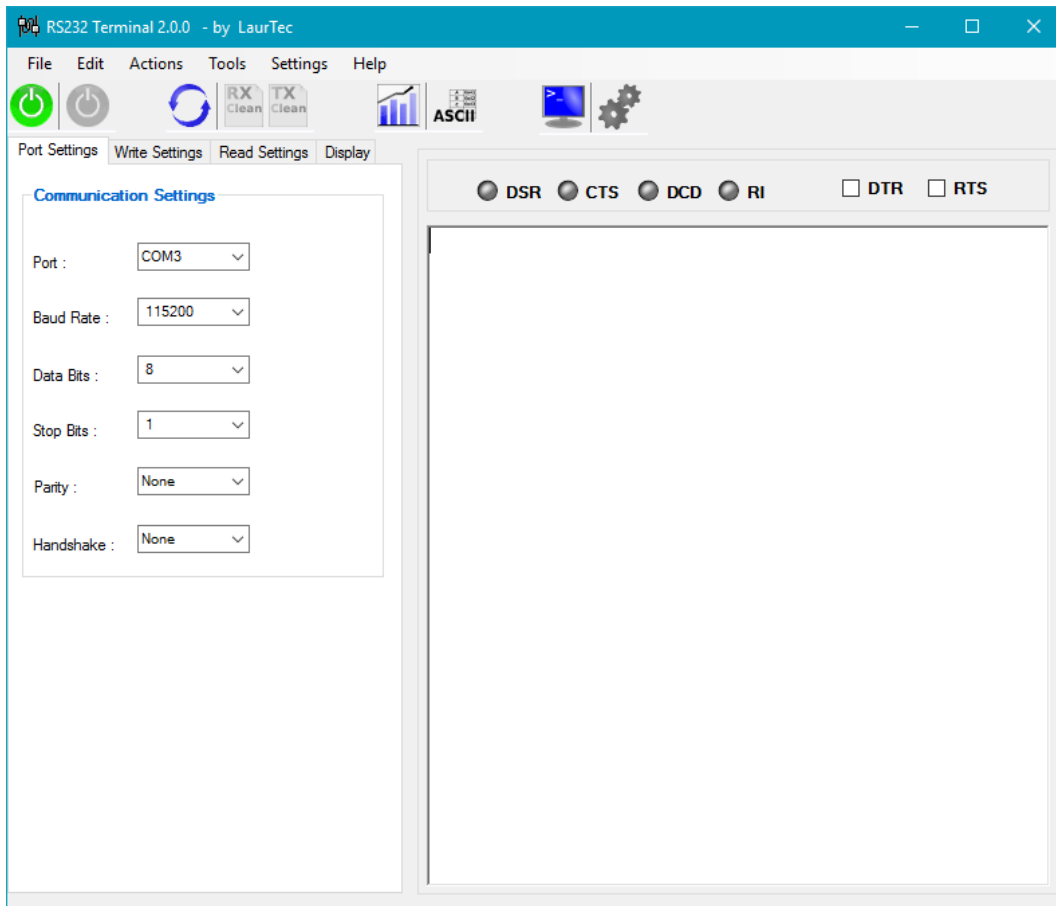


*Figure 1: First Application run.*

You can select the check box in the bottom left corner if you do not want to see that window being prompted by every execution.

## Basic Interface

The graphical user interface is quite easy and intuitive. It is made of two main areas, the setting panes on the left side and the Write/Read (TX/RX) text area on the right side, as shown in Figure 2.



**Figure 2:** Interface overview (Terminal Mode).

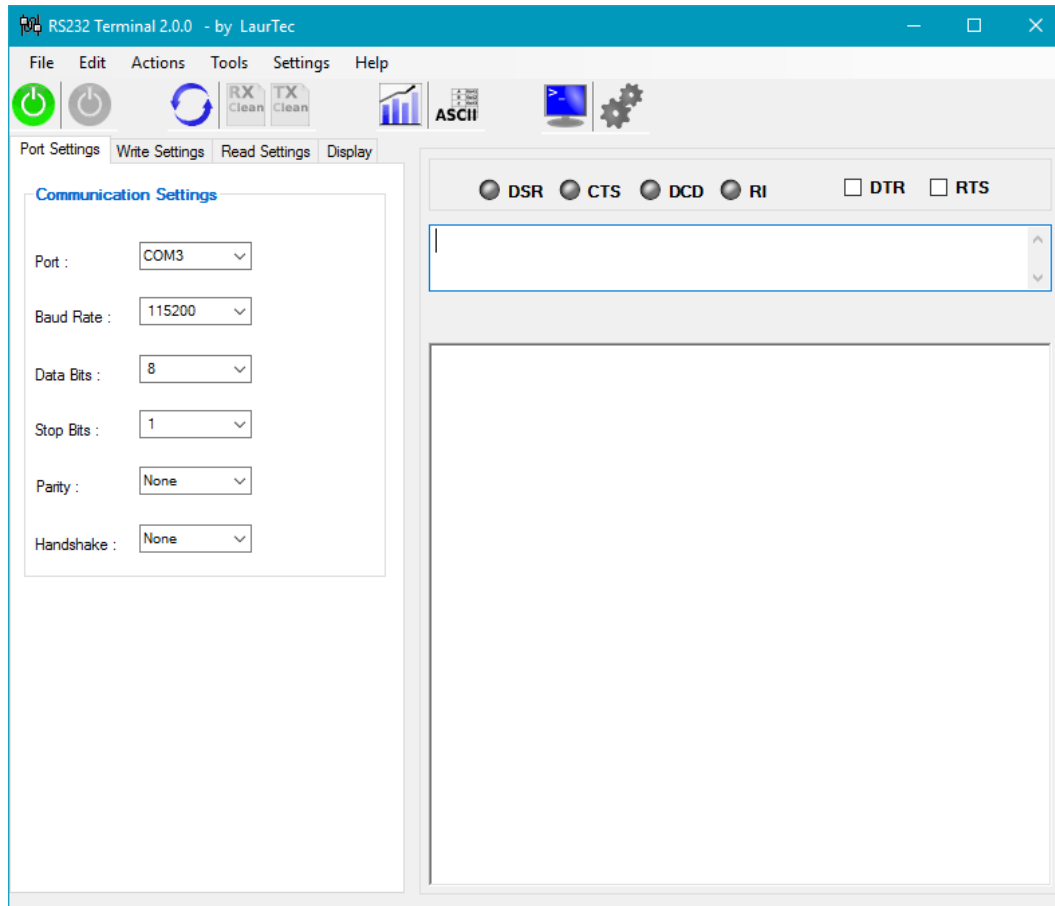
The setting panes allow several options to be enabled depending on the system requirements. The Read/Write area allows writing the data to the communication port and shows the data coming from it.

The GUI can be either set in Terminal Mode or Programmer Mode. If the Terminal Mode is active, the application behaves as a standard Terminal, so you can write ASCII characters. If the Programmer Mode is active, also the options to write the data in HEX and Integer format, beside ASCII, get enabled. You can enable one or the other modality in different ways.

- Menu Settings → Terminal Mode
- Menu Settings → Others...
- Toolbar, by pressing the blue monitor icon
- Double click on the group area just outside the Read/Write text boxes.

Since in Terminal Mode, you can write only in ASCII format, the HEX and Integer formats are disabled.

If the Programmer Mode is enabled, the GUI gets changed as shown in Figure 3. The TX/RX area get divided in two areas, TX on the top side and RX on the bottom side. Having a dedicated TX area allows more options and flexibility, which get enabled on the Write Settings pane.



**Figure 3:** Interface overview (Programmer Mode).



## Top Menu

The Top menu allows easy access to the main GUI settings and tools. The Menu does not provide access to the Port Settings, which are shown and accessible from the main window Setting panes. The Top Menu offers the following options:

### Menu File

#### **Open Log Data...**

The RX buffer can be saved within a file. This Option let you reopen log data that was previously saved. Any data log file can be opened also by using external text editors.

#### **Save Log Data...**

This function let you save the data shown within the RX buffer.

#### **Import File...**

This function let you open an ASCII file and write it directly to the TX buffer.

#### **Import Settings**

This function let you configure the GUI and port settings according to a previous setup that was saved within a configuration file.

#### **Export Settings**

This function let you export the GUI and port settings so that they can be used at later point in time. The configuration can be either opened via Import Settings or via the GUI settings. This second option allows an automated configuration at Start-up, overwriting all the default settings.

#### **Exit**

This function terminates the application and close the port in case it is still open.

### Menu Edit

#### **Undo**

This function makes a step back on any change that was done within the RX buffer text.

#### **Redo**

This function makes a step forward on any change that was done within the RX buffer text.

#### **Cut**

This function cuts the selected text within the RX buffer text.

#### **Copy**

This function copies the selected text within the RX buffer text.

#### **Paste**

This function pastes any text that was previously selected within the RX buffer text.

**Goto Line**

This function let you select a specific line within the RX buffer text. It is handy if the received data has a known format and the lines are shown within the RX buffer.

**Find Data**

This function allows searching specific data within the RX buffer. It is handy if the amount of data within the RX buffer does not allow an easy “eye searching”. The searching window allows also Regular Expressions (Regex).

**Menu Actions****Open Port**

This function opens the communication port accordingly to the selected settings.

**Close Port**

This function closes the communication port.

**Reset Port**

This function closes and reopens the selected port. It is handy during debugging, to test or recover the communication with the device under test.

**Refresh Port List**

If a new system is connected to the PC after the application got executed, to let any new port appear among the available ones, you need to refresh it.

**Clear TX Buffer**

This function clears the TX buffer text.

**Clear RX Buffer**

This function clears the RX buffer text.

**Clear Data Counters**

This function clears all the counters associated to the statistical analysis.

**Menu Tools****ASCII Table**

This function shows the ASCII table tool, which is an active table that shows the different ASCII codes with different data formats.

**Data Converter**

This function shows the Data Converter tool. This tool is a convenient way to convert data between different formats.

**Statistics**

This function shows the statistics related to the communication established with the embedded system.

**Sliders**

This function shows the Slider tool, which allows sending data by moving a slider. This is a convenient way to send data to any application where a parameter such as position, speed or a threshold, needs to be changed on the fly.

**Macros**

This function shows the Macro tool, which allows sending a fixed data pattern just by pressing a button. The tool provides several buttons that can be set to different data patterns.

**Check bits**

This function shows the Check bits tool. This tool is a convenient way to send data in binary format.

**Menu Settings**

All the settings below affect the GUI on the fly and are automatically stored within the *RS232 Terminal.cof* file. All the stored changes are automatically loaded by each application reboot.

**Terminal Mode**

This function allows to either enable the Terminal Mode or the Programmer Mode. If checked the Terminal Mode is active.

**Hide Settings Tab**

This function allows hiding the Setting panes on the left side of the GUI, any time the communication port gets activated. In this way you can get a complete monitor for the TX/RX buffer. The Setting panes are automatically shown by moving the mouse on the left side of GUI or by deactivating this option.

**Autoscroll RX Data**

This function enables the Autoscroll within the RX buffer text.

**Others...**

This function shows other settings that can be set for the GUI. All the settings are stored within the *RS232 Terminal.cof* file and are automatically loaded by any application reboot.

**Menu Help****About**

You will find my name...

**License**

It shows the license and gives you the opportunity to decide again if you want to accept it or not.

**History**

It shows all the changes through which the application have been going through over the years.

## Toolbar

Some of the options are accessible directly via the Toolbar. This allows an easy access to some of the main options and tools. The Toolbar is shown in Figure 4.



Figure 4: Toolbar.

All the Toolbar options are accessible also via the main Menu. Starting from the left side, the functions perform the following actions:

### Open Serial Port

It opens the serial port accordingly to the port settings. Once the port is opened the port settings cannot be changed. To change any setting again, you need to close the port first.

### Close Serial Port

The port is closed and the port settings can be changed to new ones.

### Refresh Port List

If you connect a new embedded system at the time the application is already running, it is possible to make it visible among the communication ports by refreshing the port list. This function is also useful in case a system is connected/disconnected from the PC and it creates a communication port via the USB CDC class.

### Clear RX Buffer Text

This function deletes the content of the RX buffer text. The icon is enabled only at the time the RX buffer has some data.

### Clear TX Buffer Text

This function deletes the content of the TX buffer text. The icon is enabled only at the time the TX buffer has some data.

### Show Communication Statistics

This function activates the Statistics window that shows the main data collected during the communication port activities.

### Show ASCII Table

This function activates the ASCII table tool that let you access the ASCII table and writes data directly out of it.

### Toggle Terminal Mode

This function toggles from the Terminal Mode to the Programmer Mode and vice versa.

### Show Settings

This function shows the GUI Settings window.

## GUI Settings

The User Interface offers several options that can be enabled to better fit different needs. The options can be accessed via the Top Menu:

| Settings → Others...

By selecting that menu, the Settings window is opened. The window offers several options, grouped in four different panes as discussed below. Any setting change is automatically stored within the file:

| RS232 Terminal.cof

The option values are written in plain text, so it is possible to open the configuration file with a standard text editor.

### Start up

The Start up pane controls the options for the license, storing the acceptance of it and the option to show it during the booting. Beside it, it offers the option to load a setting file that was previously exported via the main *RS232 Terminal* Menu:

| File → Export Settings

This setting file contains all the communication port settings and differs from the GUI settings. Importing the settings allow overwriting the default values.

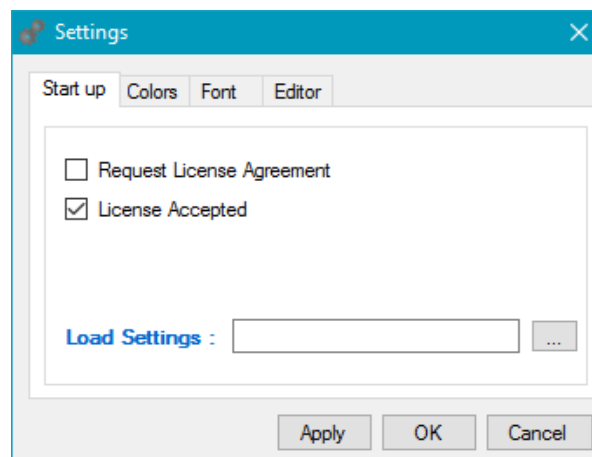


Figure 5: Start up pane.

### Colors

The Colors pane (Figure 6) allows setting the GUI from the color perspective. In particular you can set the RX text color, TX text color and the TX/RX text boxes background. The color options are some how limited but typical colors are available.

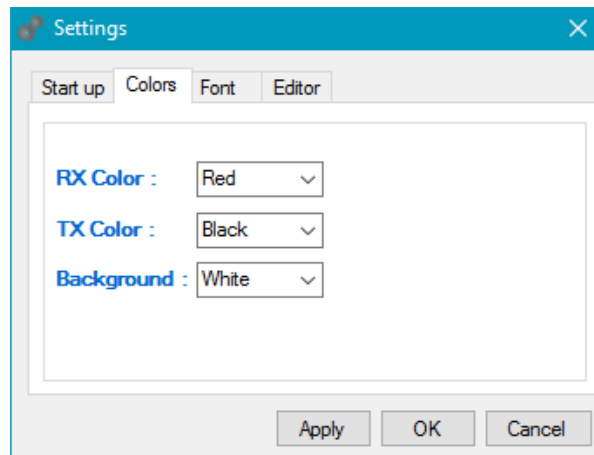


Figure 6: Colors pane.

## Font

The Font pane (Figure 7) allows setting the font type and size. These options affect both TX and RX texts.

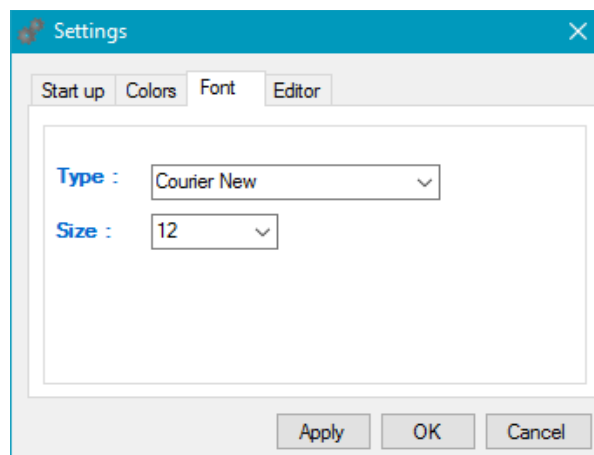


Figure 7: Font pane.

## Editor

The Editor pane has most of the options that are also available via the main *RS232 Terminal* Menu:

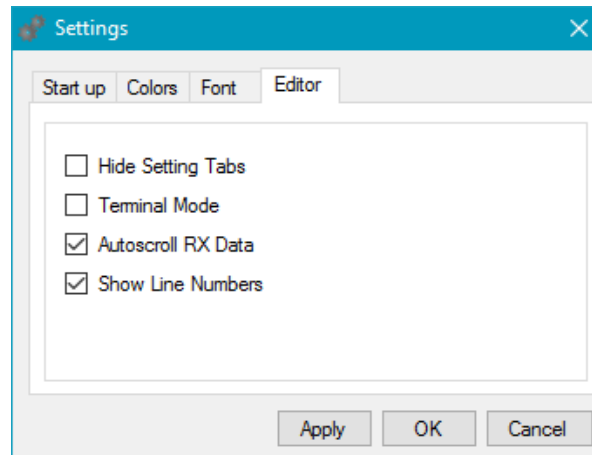
```
| Settings
```

There are some small differences between changing the settings via the Menu or via the Settings window. In the first case any change takes place right away, while on the second case you need to confirm it before the change takes effect. In both cases any change is than stored within the configuration file:

```
| RS232 Terminal.cof
```

Another difference to keep in mind is that any setting change within the Settings window,

once applied, deletes the TX and RX buffer texts. By changing only the Editor settings via the main Menu, the TX and RX buffer texts keep the data content.



**Figure 8:** *Editor pane.*

As shown in Figure 8 the Editor pane offers the following options:

### **Hide Settings Tabs**

This option, if checked, hides the Setting panes on the left side of the user interface. This is done only after the communication port gets activated. Once the settings are hidden they can appear again just by moving the mouse on the left side of the user interface or by removing the check from the main Menu or Settings window.

### **Terminal Mode**

This option, if checked, enables the Terminal Mode (Figure 2). If the option is unchecked the Programmer Mode is activated (Figure 3). Depending on the active modality, different communication options get enabled.

### **Autoscroll RX data**

This option, if checked, allows the RX text to be aligned with the last incoming data, always shown on the last line.

### **Show Line Numbers**

This option, if checked, shows the line numbers on the left side of the RX text.

## Communication Settings

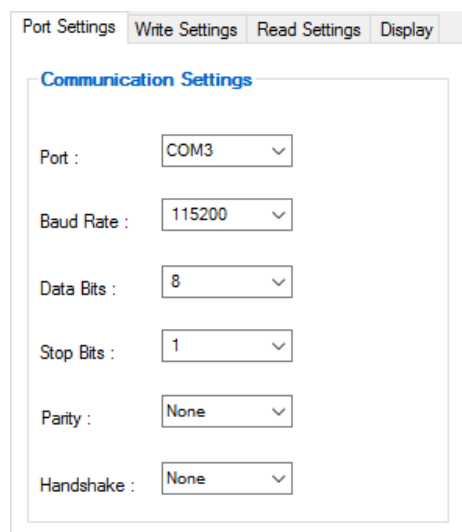
*RS232 Terminal* offers many settings for the communication between the PC and the embedded system. Those settings are all grouped within four panes: *Port Settings*, *Write Settings*, *Read Settings* and *Display*, shown on the left side of the user interface. The setting panes, once the communication port is activated, can be hidden by enabling the option:

| Hide Setting Tabs

This option can be found either in the main Menu or within the Editor Settings. In the following paragraphs are described all the details for each pane.

## Port Settings

The Port Settings pane, as shown in Figure 9, collects all the settings that can be used for the Serial Port of interest. All the fields except the field *Port* have a constant list of values, which are independent from the machine on which the software gets executed. The field *Port*, has a dynamic list that depends on the PC where the application is running on; that field, gets in fact populated on run time with all the available ports. If no ports are found the field remains empty.



**Figure 9:** Port Settings pane.

If the PC hardware configuration gets changed while the *RS232 Terminal* is running, it is possible to update the list simply by running the command *Refresh Port List* from the menu *Actions*, without the need to close and re-open the application.

A communication port can be opened from the Toolbar or from the *Actions* menu. Once the Port gets activated, the *Port Settings* pane is disabled, since no parameters can be changed on the fly.



## Write Settings

The *Write Settings* pane is divided in groups as shown in Figure 10.

The screenshot shows the 'Write Settings' pane with the following configuration:

- Data Format:** ASCII (selected), Hexadecimal, Integer.
- Send:** Clear TX buffer when message is sent (unchecked), Send as you type (unchecked), Send on Carriage Return (checked), Local Echo (unchecked), Append Carriage Return (CR) (unchecked), Append New Line Feed (LF) (unchecked).
- Enable Auto Send:** Enable Auto Send (checked), Auto Send Beep (unchecked).
- Time (ms):** 1000
- Auto-repeat:** 100
- Buffer:** Size: 1024

**Figure 10:** *Write Settings* pane.

### Data Format

The data format radio buttons allow to change the data format used while sending the data out. Depending on the format the *Send Message* text assumes a capitalization or not. The ASCII format is used as it would be for a standard text writer, so it is possible to write with capital letters or not and insert any ASCII character. Empty spaces are sent out while Carriage Return (CR) and Line Feed (LF) require special settings, since they can be masked.

The Integer format allows writing the data as an integer. The integer must be between 0-255. If multiple Integers must be sent out, they must be separated with a space (eg. 34 56 245 200).

The Hexadecimal format is the most interesting format for the embedded programmers since it allows sending the data in the format they write their codes for the embedded systems. The message can be written in any manner, grouping byte per byte separated by a space (AE 05 A3 B2), or writing it as a single word (AE05A3B2). If the system has a 16 bits core, it is possible to write the word in group of 2 bytes (AE05 A3B2). Any code smaller than 16 (decimal) must be written using a 0 in front of it. So if 9 must be sent out in Hexadecimal format you have to write 09. This rule allows writing the data either in group or all together. Any incomplete byte is not sent out. It is possible to

see from the message length, shown in the statistical data, if the written data is not complete.

**Nota**

The Hexadecimal and Integer formats are available only by selecting the Programmer Mode. If the Terminal Mode is active, those data formats are disabled.

**Send**

This group offers several options. Depending on the modality, either Terminal or Programmer mode, some options might be disabled.

**Clear TX Buffer when message is sent**

This option clears the TX buffer text any time the message is sent out.

**Send as you type**

This option sends any data written in the TX buffer text as it is typed in. All the text in the TX buffer is sent out plus the last character typed in.

**Send on Carriage Return**

This option triggers the send event, any time the Carriage return (CR) is pressed. The CR and Line Feed (LF) characters are not sent out by pressing Return. To send those codes out, you need to enable the options discussed below. In case AT commands are used, you need to remember that CR and LF are not sent out, unless enabled.

**Local Echo**

This option shows a local echo within the RX buffer text, writing it using the TX color. This lets easily differentiate the local echo from the received text.

**Append Carriage Return (CR)**

This option allows sending the CR ASCII code. Indeed the CR code is not sent out by pressing the Return button on the keyboard, unless this option is enabled.

**Append New Line Feed (LF)**

This option allows sending the LF ASCII code. Indeed the LF code is not sent out by pressing the Return button on the keyboard, unless this option is enabled.

**Auto Send**

This option is not frequently used, but when needed is doing the job of a second person. Any time you need to send repeated data and in the mean time change some settings on your board or the test equipment, this is the feature you would like to have. If you are located some meters away from the PC that is transmitting data to the embedded system and you need to regularly ping your embedded system, this is the feature you would like to have. Since this feature it may be used while you are not close to the PC, you can enable a beep, that informs you once the data is sent out. Last but not least, you

can decide the period at which the data is sent out and the number of times you want to repeat it.

## Buffer

This feature allows to change the PC buffer which is dedicated to the output data (TX).



### Nota

This number could differ from the actual one, since that parameter does not consider the buffer that the RS232 transceiver could have internally. That number specifies only the RAM that is assigned to the communication.

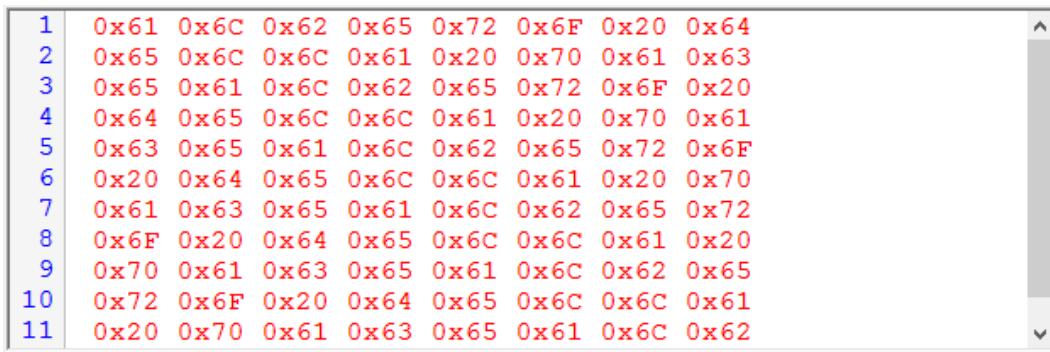
## Read Settings

The *Read Settings* pane is shown in Figure 11. Different options are grouped together by functions.

**Figure 11:** *Read Settings* pane.

## Data Parser

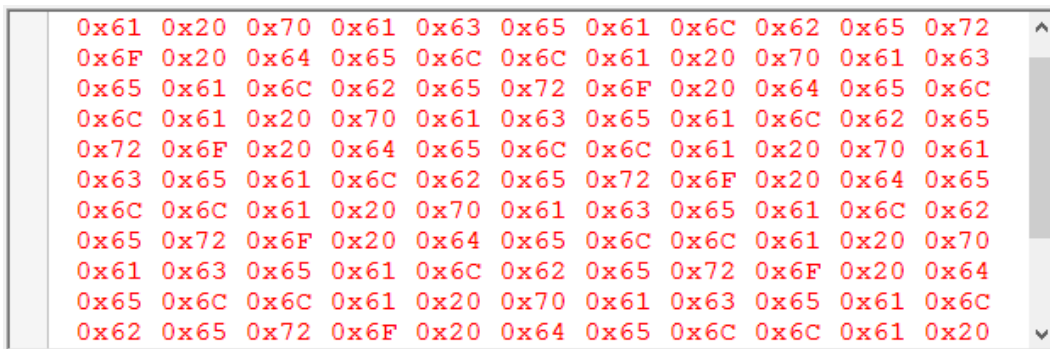
The data parser option is an handy feature that allows formatting the received data according to the embedded system data format. Many times data are sent in data packets of known length or/and with a known starting header bytes. In either case the received data can be parsed accordingly showing data in a tidy manner, making the read out easier. Each option, beside activating the Data Parsing, must be individually activated. In Figure 12 is shown an example on which the data Parsing option is set using a data packet length of 8 bytes. It's possible to see how the data get formatted inserting a carriage return any time 8 bytes are received. Enabling the option of line numbers, the data is shown in a tidy manner.



1	0x61	0x6C	0x62	0x65	0x72	0x6F	0x20	0x64
2	0x65	0x6C	0x6C	0x61	0x20	0x70	0x61	0x63
3	0x65	0x61	0x6C	0x62	0x65	0x72	0x6F	0x20
4	0x64	0x65	0x6C	0x6C	0x61	0x20	0x70	0x61
5	0x63	0x65	0x61	0x6C	0x62	0x65	0x72	0x6F
6	0x20	0x64	0x65	0x6C	0x6C	0x61	0x20	0x70
7	0x61	0x63	0x65	0x61	0x6C	0x62	0x65	0x72
8	0x6F	0x20	0x64	0x65	0x6C	0x6C	0x61	0x20
9	0x70	0x61	0x63	0x65	0x61	0x6C	0x62	0x65
10	0x72	0x6F	0x20	0x64	0x65	0x6C	0x6C	0x61
11	0x20	0x70	0x61	0x63	0x65	0x61	0x6C	0x62

**Figure 12:** Example of Data Parsing with a fixed data packet set to 8 bytes.

In Figure 13 is shown an example on which the same data is received without enabling the data parsing options.



1	0x61	0x20	0x70	0x61	0x63	0x65	0x61	0x6C	0x62	0x65	0x72
2	0x6F	0x20	0x64	0x65	0x6C	0x6C	0x61	0x20	0x70	0x61	0x63
3	0x65	0x61	0x6C	0x62	0x65	0x72	0x6F	0x20	0x64	0x65	0x6C
4	0x6C	0x61	0x20	0x70	0x61	0x63	0x65	0x61	0x6C	0x62	0x65
5	0x72	0x6F	0x20	0x64	0x65	0x6C	0x6C	0x61	0x20	0x70	0x61
6	0x63	0x65	0x61	0x6C	0x62	0x65	0x72	0x6F	0x20	0x64	0x65
7	0x6C	0x6C	0x61	0x20	0x70	0x61	0x63	0x65	0x61	0x6C	0x62
8	0x65	0x72	0x6F	0x20	0x64	0x65	0x6C	0x6C	0x61	0x20	0x70
9	0x61	0x63	0x65	0x61	0x6C	0x62	0x65	0x72	0x6F	0x20	0x64
10	0x65	0x6C	0x6C	0x61	0x20	0x70	0x61	0x63	0x65	0x61	0x6C
11	0x62	0x65	0x72	0x6F	0x20	0x64	0x65	0x6C	0x6C	0x61	0x20

**Figure 13:** Example without the Data Parsing enabled.

Beside the Packet Length, it is possible to select also the header bytes. The parser supports up to two bytes. The Start byte must be written according to the format selected in the same group, which may differ from the data format used within the Display or Write Settings panes. The Start Byte II can be enabled only if the Start byte I is enabled. The start byte options are independent from the Packet Length and they can be activated with or without it. Once the Start Byte I/II are activated, the parser ignores all the incoming bytes until the right header is received. If the Packet Length is also activated the packet has a fixed length starting with the header bytes. Once the Packet reaches the right length any exceeding byte is ignored until a new header is received.

## Auto Response

If the Parser option is activated, it is also possible to enable an Auto Response. The Auto Response is a string that has the same data format of the Write Data (selected on the Write Settings pane) and it is sent out any time the Parser finds a data match.

## Buffer

This feature allows to change the PC RX buffer RAM that is dedicated to the incoming data.

## Display

The Display pane is divided in different groups as shown in Figure 14. Several options allow formatting the data in the RX buffer text in special ways. These options are handy if post data processing is needed and the end application needs a special data formatting. The meaning of many options is quite straightforward.

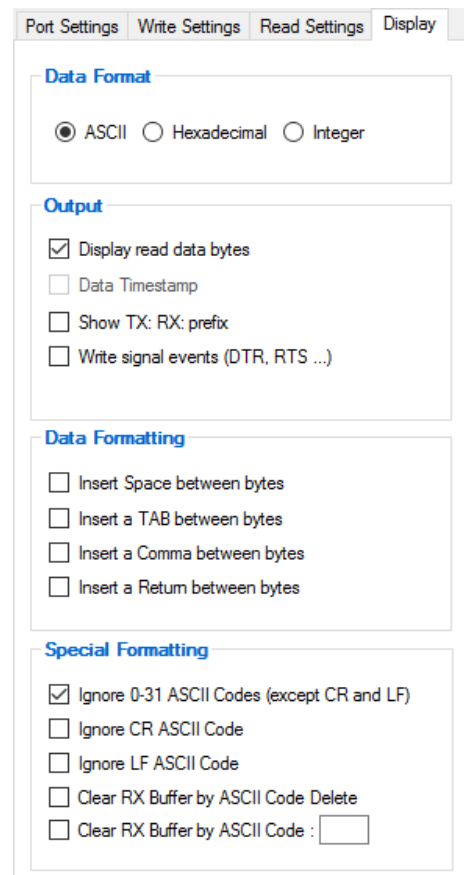


Figure 14: Display pane.

### Data Format

Data format is quite an handy feature that let you adapt the text according to the format you are using for sending the data out. The data that is shown in the *Read Messages* box have the format according to the one selected in that group.

### Output

The group offers the following options:

#### Display read data bytes

Enabling the check, shows all the received data into the Read Messages box. Disabling it will not show any further data into the RX buffer text.

#### Data Timestamp

Enabling this option writes date and time in the RX buffer text as shown below. This

option works in Programmer mode only.

```
| RX : 22/04/2017 11:06:50: Test
```

### Show TX: RX: prefix

Enabling this option shows the TX and RX prefixes as shown below. This option works in Programmer mode only.

```
| TX : Test  
| RX : Test
```

### Write signal events (DTR, RTS...)

This option writes the occurrence of any RS232 signal event in the RX buffer text. If it is not enabled, the events are shown only by the Status LEDs, located on the top of the user interface. The events are written as the example below:

```
| TX : DTR is True  
| TX : DTR is False  
| TX : RTS is True
```

## Data Formatting

At first glance you might underestimate how important these features are. Any time you receive a byte you can choose what to insert before the one that will come next. This is a quite useful feature for post data processing. For instance if your device sends temperature data over time, it might be useful inserting the Carriage Return, getting all the data in one column. If you have multiple values sent at the same time, such as temperature (°C), humidity (%), pressure (atm), you may consider adding a Comma to Separate the Values (CSV format) and enable the parser at fixed data length. An example is shown below:

```
| 026, 057, 001,  
| 027, 059, 001,  
| 027, 055, 001,
```



### Nota

Many data analyzers allow as data separator also the space and tabulation, beside the comma.

## Special Formatting

This group of options allow ignoring some data. In particular the ASCII character 0-31 do not have a printed symbol, so you may want to ignore it. Other characters that might be ignored depending on the application, are the CR (Carriage Return) and LF (Line Feed). Beside ignoring some ASCII code, the special formatting has two other options:

- Clear RX Buffer by ASCII code Delete
- Clear RX Buffer by ASCII code: xx

The first option clears the RX buffer text any time the embedded system sends the Delete code (0x18). This is useful in case the embedded system sends a menu via the terminal and moving from one menu to another requires removing the previous one.

The second option is like the first, but allows setting a custom character value for the monitor clean up. The code must be written as integer value.

## RS232 Signals

The RS232 protocol supports, beside the TX and RX lines, several acknowledgment signals (ACK). Those signals are mostly handled via Hardware, if activated on the Port Settings. Most of the embedded systems support raw communication without ACK signals, nevertheless some of those signals are sometimes used for other applications and it is handy to know if a certain event took place or not. The DSR, CTS, DCD, RI signals are input signals for the PC, while the the DTR and RTS signals can be controlled via the GUI as output signals. *RS232 Terminal* provides a graphical interface to monitor the incoming signals. Figure 15 shows the LEDs that get turned on (green color) any time there is an event.

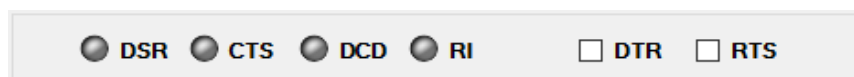


Figure 15: LEDs pane.

The check boxes allow a direct control of the DTR and RTS output lines.

Since the incoming signals might be relatively fast, *RS232 Terminal*, beside the graphical visualization, offers a data event logging via the RX buffer text. Indeed, you can activate a message at any occurrence of any event, either from the input or output signals. This can be done via the Display Settings pane, activating the option:

```
| Write signal events (DTR, RTS...)
```

## Debugging Tools

Debugging embedded systems via a Terminal may require additional tools that allow things beyond just writing ASCII codes. *RS232 Terminal* provides several tools that are not typically included in any Terminal application and normally require writing a dedicated application for it. The tools available at this time are:

- ASCII Table
- Data Converter
- Statistics
- Sliders
- Macros
- Check bits

In the todo list there are also other tools that might be added at a later point in time.

The last three tools (Sliders, Macros, Check bits) can be opened as multiple instances just by clicking multiple times on the Menu. This allows more flexibility by controlling the embedded systems. All the tools are independent from each other so beside having multiple instances of each, you can have it running in parallel.

Each tool, has a Top Menu Settings with the option:

| `Keep the Window on Top`

enabling the option keeps the window always in foreground (top).



## ASCII Table

The ASCII table simply shows all the ASCII codes, as shown in Figure 16. In particular for each ASCII code it is shown the equivalence between Decimal, Hexadecimal and Character values. Beside that, if the check box on the bottom left side (send the code by clicking on it) is enabled, by selecting a code, the value is sent out automatically in the right format selected in the Write Settings pane. Depending on the GUI modality, the data might be sent out straightaway or not.

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0x00	NULL	32	0x20	Space	64	0x40	@	96	0x60	`
1	0x01	Start of Heading	33	0x21	!	65	0x41	A	97	0x61	a
2	0x02	Start of Text	34	0x22	"	66	0x42	B	98	0x62	b
3	0x03	End of Text	35	0x23	#	67	0x43	C	99	0x63	c
4	0x04	End of Transmit	36	0x24	\$	68	0x44	D	100	0x64	d
5	0x05	Enquiry	37	0x25	%	69	0x45	E	101	0x65	e
6	0x06	Acknowledge	38	0x26	&	70	0x46	F	102	0x66	f
7	0x07	Audible Bell	39	0x27	'	71	0x47	G	103	0x67	g
8	0x08	Backspace	40	0x28	(	72	0x48	H	104	0x68	h
9	0x09	Horizontal Tab	41	0x29	)	73	0x49	I	105	0x69	i
10	0x0A	Line Feed	42	0x2A	*	74	0x4A	J	106	0x6A	j
11	0x0B	Vertical Tab	43	0x2B	+	75	0x4B	K	107	0x6B	k
12	0x0C	Form Feed	44	0x2C	,	76	0x4C	L	108	0x6C	l
13	0x0D	Carriage Return	45	0x2D	-	77	0x4D	M	109	0x6D	m
14	0x0E	Shift Out	46	0x2E	.	78	0x4E	N	110	0x6E	n
15	0x0F	Shift In	47	0x2F	/	79	0x4F	O	111	0x6F	o
16	0x10	Data Link Escape	48	0x30	0	80	0x50	P	112	0x70	p
17	0x11	Device Control 1	49	0x31	1	81	0x51	Q	113	0x71	q
18	0x12	Device Control 2	50	0x32	2	82	0x52	R	114	0x72	r
19	0x13	Device Control 3	51	0x33	3	83	0x53	S	115	0x73	s
20	0x14	Device Control 4	52	0x34	4	84	0x54	T	116	0x74	t
21	0x15	Neg. Acknowledge	53	0x35	5	85	0x55	U	117	0x75	u
22	0x16	Synchronous idle	54	0x36	6	86	0x56	V	118	0x76	v
23	0x17	End Trans. Block	55	0x37	7	87	0x57	W	119	0x77	w
24	0x18	Cancel	56	0x38	8	88	0x58	X	120	0x78	x
25	0x19	End of Medium	57	0x39	9	89	0x59	Y	121	0x79	y
26	0x1A	Substitution	58	0x3A	:	90	0x5A	Z	122	0x7A	z
27	0x1B	Escape	59	0x3B	;	91	0x5B	[	123	0x7B	{
28	0x1C	File Separator	60	0x3C	<	92	0x5C	\	124	0x7C	
29	0x1D	Group Separator	61	0x3D	=	93	0x5D	]	125	0x7D	}
30	0x1E	Record Separator	62	0x3E	>	94	0x5E	^	126	0x7E	~
31	0x1F	Unit Separator	63	0x3F	?	95	0x5F	_	127	0x7F	ï

Send the code by clicking on it

Figure 16: ASCII Table.

If the Terminal Mode is selected, the data is sent directly out. If the Programmer Mode is selected, depending on the settings, the data might be added within the TX text or sent directly out.

## Data Converter

While writing data, it might be handy having a conversion tool to convert the data from one format to another. The Data Converter tool, shown in Figure 17, offers this option. Beside converting the data it offers also the option to send the converted data directly to the TX buffer. Depending on the transmission options, the data might be sent out directly or added to the TX buffer text first.

The input data format is selected on the left side while the output one is selected on the right side. Within the text on the left you can enter the value to be converted, while on the right text box it is shown the converted data.

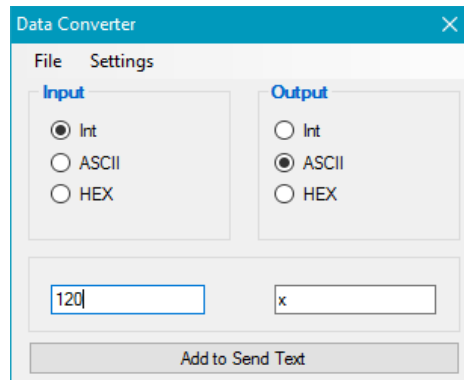


Figure 17: Data Converter tool.

## Statistics

Any time a communication between the PC and an embedded system takes place, it could be useful to know how much bytes have been sent or received and some special events occurrences. The Statistics tool offers this feature, providing in a table, some of the most relevant data.

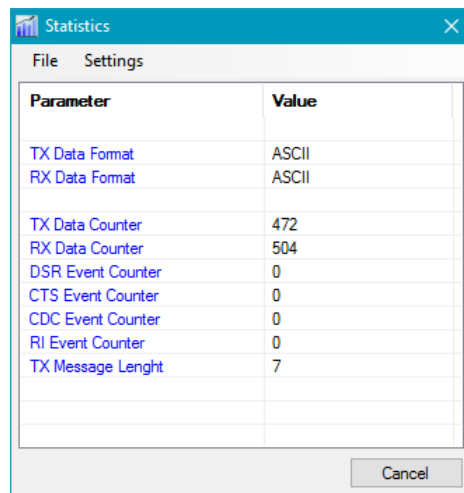


Figure 18: Statistic tool.

All the counters can be restarted by pressing the option in the main *RS232 Terminal* Menu:

| Actions → Clear Data Counters

## Sliders

Embedded systems that implement control loops or require thresholds, may benefit from the Sliders tool shown in Figure 19. This tool allows sending a byte which value depends from a slider position. There are 4 sliders that can be enabled individually. Each one can have a different range of values, defined by a minimum and maximum one (0-255). The field *Value* defines the current position (read only value).

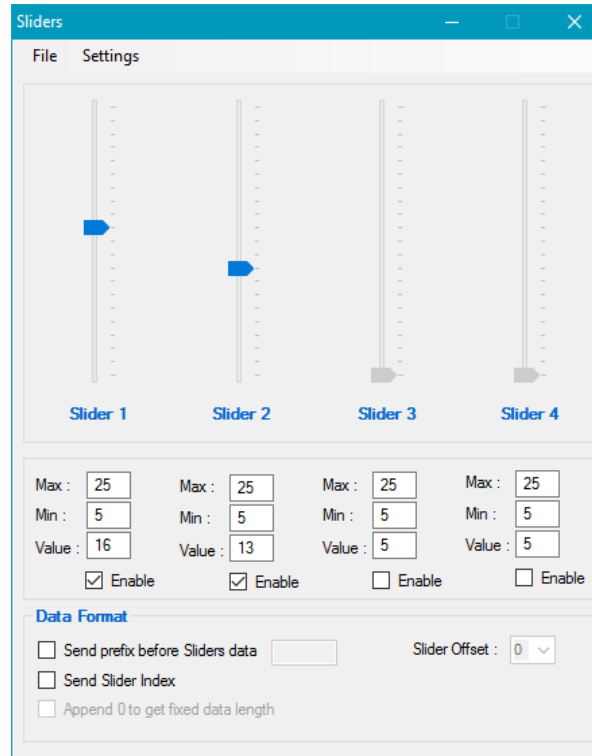


Figure 19: Slider tool.

The slider values must be defined as integer but the data can be sent out as either ASCII or Integer, depending on the Write Settings pane. If the format is ASCII the number is divided in digits and are sent as ASCII value. For example the number 23 is sent out with two characters '2' and '3'. If the ASCII format is selected it is possible to append a 0 to get a fixed data length, defined by the maximum value. So if you have to send 3 and the maximum value is 25, the application sends out 03. If the maximum value is 100, it sends 003. Among the other data format options, there is the option to send an header before the slider values. The header data format must conform either the ASCII or Integer format selected on the Write Settings pane. Before the slider value, it is possible to send out also the slider's index. The first slider has index 1, while the last one has index 4. Since it is allowed to run multiple instances of the Slider tool, it is possible to set an offset to be added to the base index. While it is possible to open as many instances as needed, the offset supports up to 4 windows.

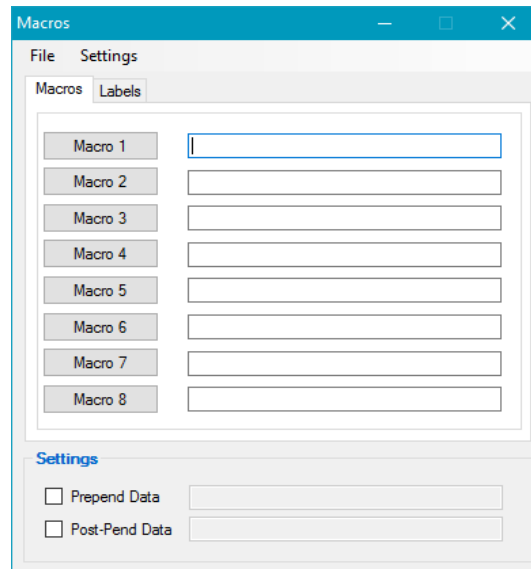


### Nota

Each slider name can be changed by clicking on the Slider name and then pressing return once the name has been changed.

## Macros

Whenever it is required to send specific commands or data sequences depending on an actions to perform, it might be handy if those commands are stored and set within a button. This is what the tool Macros does. In Figure 20 is shown the main window with empty data strings.



**Figure 20:** *Macros tool.*

Each string can be initialized according to the data format which is set within the Write Settings pane. By pressing each button the associated string is sent out. To properly fit the window with the end application, you can rename the buttons simply by changing the names within the Labels pane. All the settings can be stored within an external file and reloaded as needed. This can be done via the Menu:

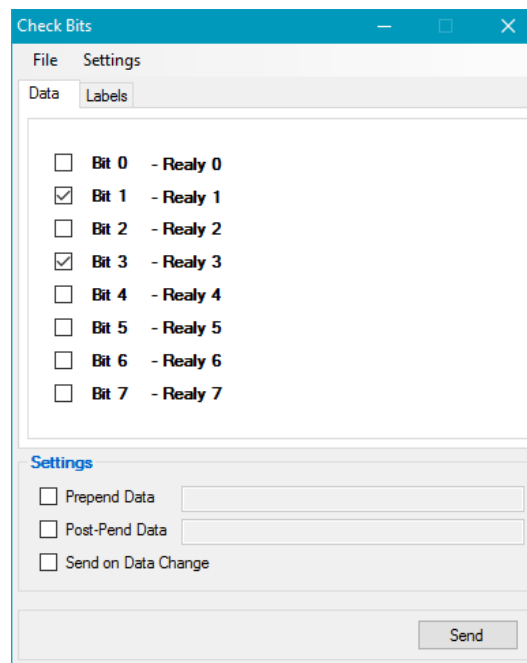
| File → Export Settings

To farther trim the data to the specific application, it is also possible to define an header data string that can be sent before or after each Macro. These options can be enabled via the check boxes in the bottom left corner.

## Check bits

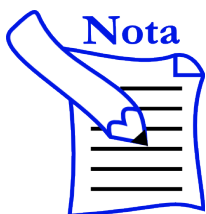
Any time it is required to turn ON and OFF a certain peripheral it could be handy sending the data in binary form, especially if each register bit refers to a special function. The Check Bits tool, shown in Figure 21, offers this feature. Each byte is divided in bits, so it is possible to access it without the need of any conversion. The tool has two panes: Data and Labels. The Data pane is where you can access each bit while the Labels pane is where you can set a bit name/description, reflecting the application specifications. All the settings and descriptions can be stored within a file via the Menu:

| File → Export Settings



**Figure 21:** *Check Bits tool.*

On the bottom side of the window there are the options to activate a data header before or after the byte that reflects the checked bits (Hexadecimal format). This is useful if the checked bits are part of a command. The header bytes may represent the command, while the checked bit values can be changed. The tool also offers the option to send the data on any check bit change, rather than sending it by pressing the button *Send*.



The Check Bits tool works with Hexadecimal format. By running it, the data format is changed to Hexadecimal.

## Alphabetical index

### A

About.....	11
ACK.....	23
acknowledgment.....	23
acknowledgment signals.....	23
Actions.....	16
Append Carriage Return (CR).....	18
Append New Line Feed (LF).....	18
ASCII.....	7, 17, 23 e segg.
ASCII codes.....	25
ASCII table.....	12
ASCII Table.....	10, 24 e seg.
AT commands.....	18
Auto Response.....	20
Auto Send.....	18
Autoscroll RX data.....	15
Autoscroll RX Data.....	11

### B

beep.....	18
binary format.....	11
Buffer.....	19 e seg.

### C

Carriage return.....	18
Carriage Return.....	23
CDC.....	4
Check bits.....	11, 24
Check Bits.....	29
Clear Data Counters.....	10, 26
Clear RX Buffer.....	10
Clear RX Buffer by ASCII code.....	23
Clear RX Buffer Text.....	12
Clear TX Buffer.....	10
Clear TX Buffer Text.....	12
Clear TX Buffer when message is sent.....	18
Close Port.....	10
Close Serial Port.....	12
Colors.....	13
Communication Device Class.....	4
Communication Settings.....	16
Copy.....	9
CR.....	18, 23
CSV format.....	22
CTS.....	23
Cut.....	9

### D

Data Converter.....	10, 24, 26
data format.....	17
Data Format.....	21
Data Format .....	17

Data Formatting.....	22
Data Parser.....	19
Data Parsing.....	19
Data Timestamp.....	21
DCD.....	23
Display.....	16, 21
Display read data bytes .....	21
Display Settings.....	23
DSR.....	23
DTR.....	23

### E

Editor.....	14
Editor Settings.....	16
Exit.....	9
Export Settings.....	9, 13

### F

Find Data.....	10
Font.....	14

### G

Goto Line.....	10
----------------	----

### H

Hardware Requirements.....	5
header bytes.....	20
HEX.....	7
Hexadecimal format.....	29
Hide Settings Tab.....	11
Hide Settings Tabs.....	15
History.....	11
HyperTerminal.....	4

### I

Import File.....	9
Import Settings.....	9
Integer.....	7

### K

Keep the Window on Top.....	24
-----------------------------	----

### L

Labels.....	29
LEDs.....	23
LF.....	18, 23
License.....	11
license agreement.....	6
Line Feed.....	18, 23
Local Echo.....	18

### M

Macros.....	11, 24, 28
Microprocessor.....	5

### O

Open Log Data.....	9
Open Port.....	10

Open Serial Port.....	12	Serial Port.....	16
Operative System.....	5	Show ASCII Table.....	12
Others.....	11	Show Communication Statistics .....	12
Output.....	21	Show Line Numbers.....	15
<b>P</b>		Show Settings.....	12
Packet Length.....	20	Show TX: RX: prefix.....	22
parser.....	22	Sliders.....	11, 24, 27
Parser.....	20	Software Installation.....	6
Paste.....	9	Special Formatting.....	23
Port.....	16	Start byte.....	20
Port Settings.....	16	Start byte I.....	20
Programmer Mode.....	7, 25	Start Byte II.....	20
<b>R</b>		Start up.....	13
RAM.....	19	Statistics.....	10, 24, 26
Read Messages box.....	21	Status LEDs.....	22
Read Settings.....	16, 19	<b>T</b>	
Redo.....	9	Terminal Mode.....	7, 11, 15, 25
Refresh Port List.....	10, 12, 16	Toggle Terminal Mode.....	12
Regex.....	10	Toolbar.....	12
Regular Expressions.....	10	<b>U</b>	
Reset Port.....	10	UART.....	4
RI.....	23	Undo.....	9
RS232.....	4	USB.....	4
RS232 emulation.....	4	USB CDC Class.....	4
RS232 Terminal.cof.....	11, 13	User Interface.....	13
RS422.....	4	<b>W</b>	
RS485.....	4	Windows 10.....	5
RTS.....	23	Windows 7.....	5
<b>S</b>		Windows XP.....	4
Save Log Data.....	9	Write Settings.....	16 e seg.
Send.....	18	Write signal events (DTR, RTS.....)	22
Send as you type.....	18	<b>.</b>	
Send on Carriage Return.....	18	.NET Framework.....	5

## Bibliography

[1] [www.LaurTec.it](http://www.LaurTec.it): It is the official web page where you can download the latest version of the software *RS232 Terminal*.



## History

Date	Version	Author	Description
05/05/17	2.0	Mauro Laurenti	New documentation for the software version 2.0.0
18/09/10	1.0	Mauro Laurenti	Original version