

User Manual

GTR-01

A Quantum, Two-Channel, 2.2 Mb/s,
USB True Random Number Generator

Rev. 2.3 – 17 Apr 2024



BENEV[®]

The specifications of the products described in this document are subject to change without prior notice. Both the software and documentation are copyrighted with all rights reserved by *Benev Science & Technology Ltd.* Neither the whole nor any part of the information contained in, or the products described in this manual, may be adapted or reproduced in any material or electronic form without the prior written consent of the copyright holder.

Copyright © 2024 Benev Science & Technology Ltd. All rights reserved.

BENEV® and *StudioGTR™* are trademarks or registered trademarks of *Benev Science & Technology Ltd.*

Other brand and product names referred to are trademarks or registered trademarks of their respective owners.




Benev Science & Technology Ltd.

Phone: +359 89 923 0345


Website: www.benev.biz


141 Tzar Simeon Veliki
Stara Zagora 6000,
Bulgaria

Used symbols

Symbol	Meaning
	Action related to software performance
 NOTE:	Supplementary information
 CAUTION:	Important information

Safety

 **NOTE:** Read this manual before using GTR-01 random number generator.

 **CAUTION:** Do not expose this device to impacts beyond the limits specified in the documentation! Use this instrument in conjunction only with products that are in compliance with the quality and safety standards.

Warranty

Benev Science & Technology Ltd. warrants that this product will be free from defects in materials and workmanship for a period of two (2) years from the date of shipment. If any such product proves defective during this warranty period, *Benev Science & Technology Ltd.*, at its option, will repair the defective product without charge for parts and labor, or will replace it.

It is customer's responsibility to notify us in the event of a defect occurring within the warranty period, as well as packing preparation and payment of the transportation costs to the our office. *Benev Science & Technology Ltd.* bears the transportation costs of delivering the repaired or replaced product back to the customer.

This warranty shall not apply to any defect, failure or damage caused by improper use or improper or inadequate maintenance and care. *Benev Science & Technology Ltd.* shall not be obligated to furnish service under this warranty in the following cases:

- The product has been disassembled, modified, or repaired by persons who have not been explicitly authorized for this by *Benev Science & Technology Ltd.* and in the event of damage or replacement of warranty stickers.
- The product has been under impacts beyond the limits specified in its documentation or has been used in conjunction with defective products or such that are not in compliance with the quality and safety standards.
- In the case of natural disaster, such as fire, earthquake, flood, etc.

Table of contents

Used symbols	iii
Safety	iv
Warranty	v
List of figures	vii
List of tables	vii
1. Overview	1
1.1 Basic features	1
1.2 Applications	1
2. GTR-01	2
2.1 Device overview	2
2.2 Noise generating process	3
2.3 Basic scheme	4
2.4 Technical specifications	4
2.5 Driver installation	5
3. StudioGTR	8
3.1 System requirements	8
3.2 Application installation	8
3.3 Main screen	9
3.4 StudioGTR basic features	9
3.5 Tools	11
3.5.1 Visualizer	11
3.5.2 Integer Numbers	12
3.5.3 Float Numbers	13
3.5.4 Strings	14
3.5.5 Random Noise	16
3.5.6 Bitmaps	17
3.5.7 Lottery Numbers	18
3.5.8 Random Event Logger	19
3.5.9 Statistical Tests	20
3.5.10 Calibration	21
3.6 Settings	22
3.6.1 Channel	22
3.6.2 Transfer	22
3.6.3 Pseudorandom Generator	23
3.7 Multilingual support	24
3.8 Application preferences	25

4. Low level API	26
4.1 GTR-01 IO commands	26
4.1.1 GET_RND_CHANNEL_A	27
4.1.2 GET_RND_CHANNEL_A_CALIBRATION	28
4.1.3 GET_RND_CHANNEL_B	28
4.1.4 GET_RND_CHANNEL_B_CALIBRATION	28
4.1.5 GET_RND_NORMAL	28
4.1.6 GET_RND_NORMAL_PLUS	29
4.1.7 GET_RND_NORMAL_DEBIAS	29
4.1.8 GET_RND_CHANNEL_A_ASYNC	29
4.1.9 GET_RND_CHANNEL_A_CALIBRATION_ASYNC	29
4.1.10 GET_RND_CHANNEL_B_ASYNC	30
4.1.11 GET_RND_CHANNEL_B_CALIBRATION_ASYNC	30
4.1.12 GET_RND_NORMAL_ASYNC	30
4.1.13 GET_RND_NORMAL_PLUS_ASYNC	30
4.1.14 GET_RND_NORMAL_DEBIAS_ASYNC	31
4.1.15 SET_CALIBRATION	31
4.1.16 GET_CALIBRATION	31
4.1.17 GET_DEVICE_ID	32
Appendix A: Statistical tests	33
Appendix B: FTDI IO Methods Wrapper in C#	34
Appendix C: GTR-01 IO Example	37
Index	40

List of figures

2.1	GTR-01 (top view)	2
2.2	GTR-01 Front panel	2
2.3	Reverse-biased p-n junction	3
2.4	V-I characteristic of p-n junction	3
2.5	GTR-01 basic scheme	4
2.6	Driver installation – step 1	5
2.7	Driver installation – step 2	5
2.8	Driver installation – step 3	6
2.9	Driver installation – step 4	6
2.10	Driver installation – step 5	6
2.11	Driver installation – step 6	6
2.12	Successful driver installation	7
3.1	StudioGTR main screen	9
3.2	Visualizer – screen	11
3.3	Integer Numbers – screen	12
3.4	Float Numbers – screen	13
3.5	Strings – screen	14
3.6	Random Noise – screen	16
3.7	Bitmaps – screen	17
3.8	Lottery Numbers – screen	18
3.9	Random Event Logger – screen	19
3.10	Statistical Tests – screen	20
3.11	Calibration – screen	21
3.12	Synchronous mode of operation	22
3.13	Asynchronous mode of operation	23
3.14	Preferences dialog box	25

List of tables

1.1	Basic features	1
2.1	Technical specifications	4
3.1	System requirements	8
4.1	List of IO commands	27
A.1	Statistical tests	33

1. Overview

GTR-01 is a hardware system generating true random numbers. The innovative multiplatform design, convenient USB interface, the high data rate of up to 2.2 Mb/s and the compact size of GTR-01, make it the right choice for many applications related to cryptography and security, scientific research, computer simulations, modeling, lotteries, gambling, applied mathematics and statistics, etc. In many of the above areas, it is important to use a source of random numbers based on a physical phenomenon, unlike the computer-generated pseudo random numbers using a known and predictable numerical algorithm.

The GTR-01 is developed with a vision for the requirements of modern hardware design, providing security and functional reliability on the one hand and a sufficiently simplified and optimized software interface for high-speed communication with the PC, on the other. The work of GTR-01 is based on a physical phenomenon called avalanche noise in the reverse-biased p-n junction and is quantum in its nature, indeed. That makes it an appropriate choice as the source of entropy in hardware random number generators.

1.1 Basic features

GTR-01 Features
Quantum phenomenon as a physical source of entropy
Two completely independent analog channels with software XOR function
12-bit precise calibration on every channel with EEPROM values storage
Data rate up to 2.2 Mb/s random bits through a standard USB interface
Multiplatform design including Windows, Linux, Mac OS and Android
Programming API including GUI application, examples and low-level drivers
Statistically proven output data stream quality
Compact and robust aluminum shielded enclosure

1.2 Applications

- Cryptography and security
- Lotteries and gambling
- PIN generation
- Scientific research
- Computer simulations
- Statistics

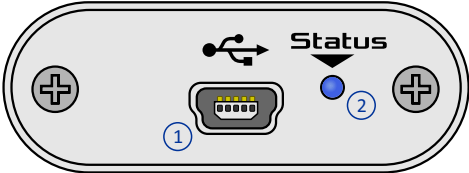
2. GTR-01

2.1 Device overview

GTR-01 (top view)



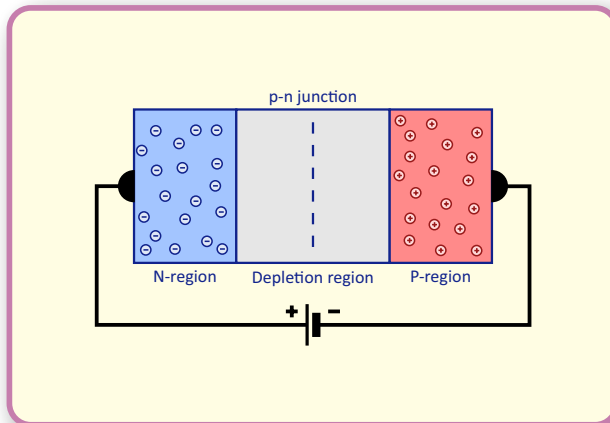
GTR-01 Front panel



- 1. USB mini connector
- 2. Status LED

2.2 Noise generating process

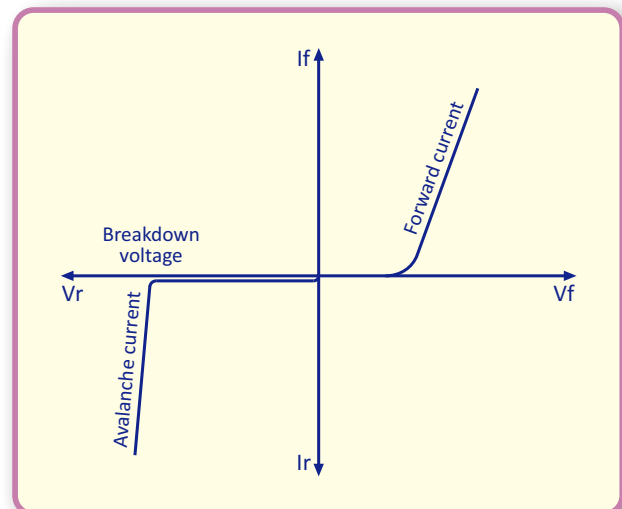
GTR-01 uses a physical phenomenon known as avalanche noise in the reverse-biased p-n junction as a primary source of entropy.



Reverse-biased p-n junction.

The generated noise in this process has a complicated structure. It consists primarily of quantum shot noise with the addition of the avalanche excess noise as a consequence of the internal amplification. The reverse-biased voltage at which described avalanche process occurs is known as a breakdown voltage.

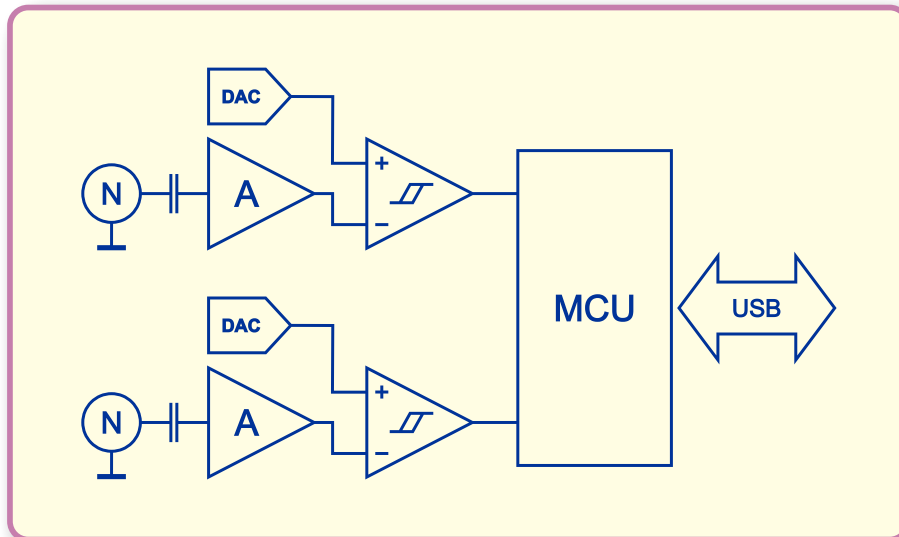
Under a reverse-bias voltage, the p-n junction's depletion region widens. Thus, a high-strength electric field arises across the junction. Unbound by random thermal fluctuations, negative charge carriers (electrons) and positive ones (holes) will move towards the positive and negative electrodes, respectively. If the electric field is strong, the electrons and holes will be accelerated to high enough speeds to knock other bound electrons free which will be accelerated too and so on, thus creating an avalanche and increasing the current.



V-I characteristic of p-n junction.

2.3 Basic scheme

The basic device scheme is shown below.



GTR-01 basic scheme.

GTR-01 has two independent analog channels. In every channel, the signal from the physical noise source is amplified to the level suitable for detection and is connected to the one input of a high-speed analog comparator. The other comparator's input is fed by a 12-bit digital-to-analog converter and is used to precisely calibrate the channel. Analog comparators outputs connect to the MCU and the 0's and 1's are further processed. Communication of the GTR-01 with the PC is through a USB interface.

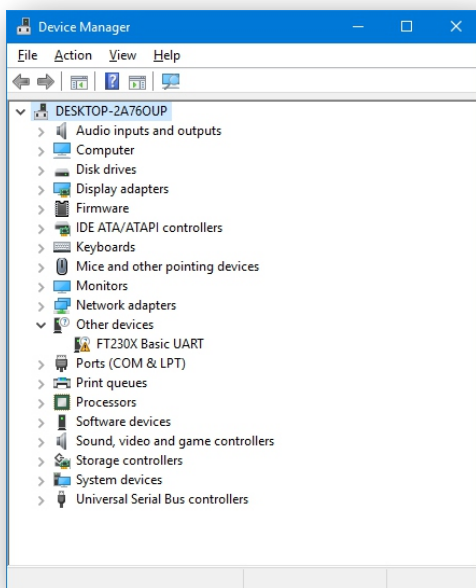
2.4 Technical specifications

Specifications	
Noise source	Avalanche noise, reverse-biased p-n junction
Number of independent channels	2
XOR	Yes
Max bit rate	2.2 Mb/s
Operating temperature range	0 ÷ 50 °C
Platform	Windows, Linux, Mac OS, Android
Dimensions	74 x 41 x 15 mm
Weight	50 g

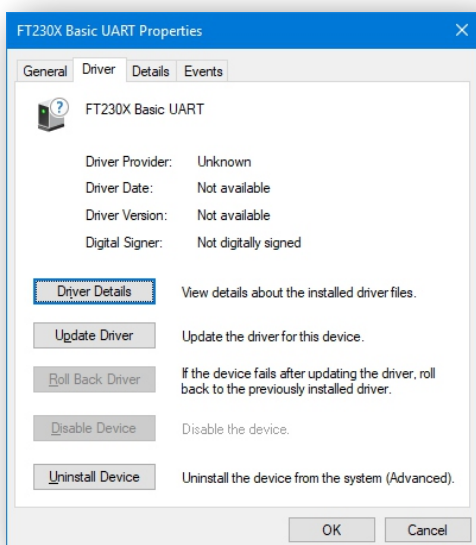
2.5 Driver installation

For proper operation, the GTR-01 need the USB driver pack to be installed. You can download drivers from the tech info section on the product's page: www.benev.biz/en/quantum-random-generators.html

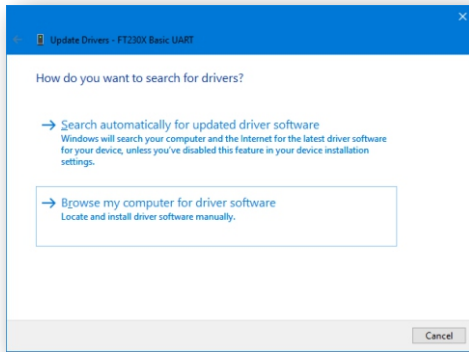
The example below demonstrates the driver installation procedure on Windows 10. For other platforms, the installation follows similar steps, but you need to use a proper driver pack.



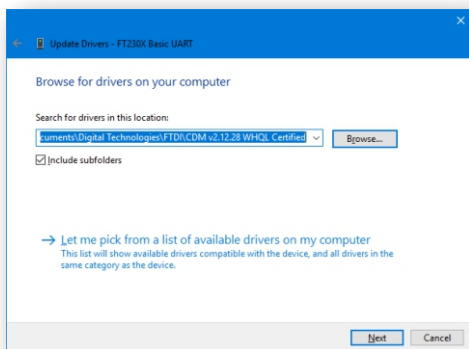
Step 1. Connect GTR-01 to the PC. When device is connected for the first time, Windows automatically installs some driver by default, but this is not the correct one. In the Device Manager the device is displayed with a yellow warning mark, as is shown on the left.



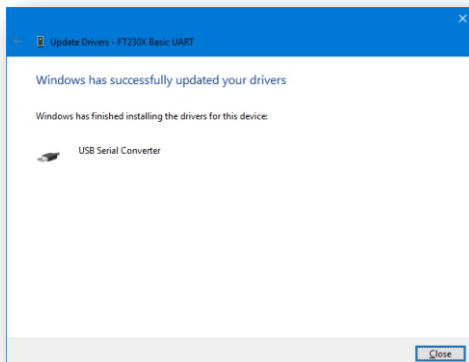
Step 2. In the Device Manager right-click on the device and choose Properties from the drop-down menu. In the Driver tab click Update Driver button.



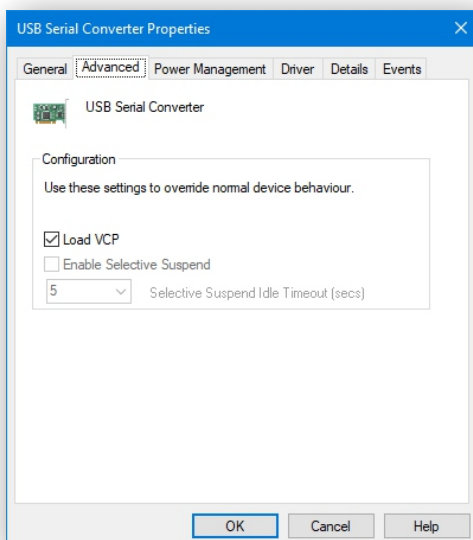
Step 3. In the next window choose Browse my computer for driver software.



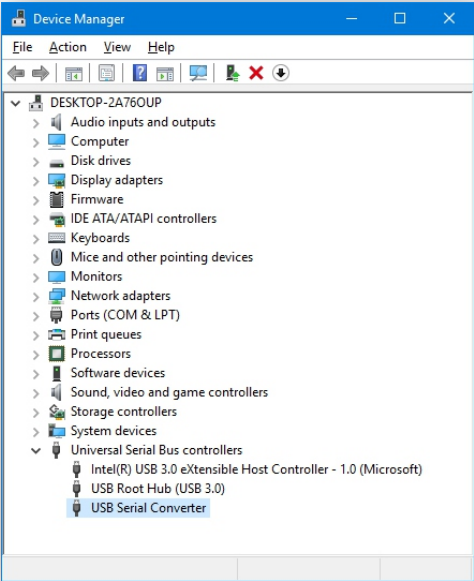
Step 4. Click the Browse button and navigate to the folder containing the driver pack.



Step 5. Wait just a moment for the driver to be installed successfully.



Step 6. Optionally, after successful driver installation, in the Device Manager, right-click on the device and choose Properties from the drop-down menu. In the Advanced tab, you can choose whether to load the Virtual COM Port driver and communicate with GTR-01 as a serial COM Port device.




After successful driver installation, GTR-01 is displayed in the Device Manager, under the USB controllers node, as an properly working USB Serial Converter.

3. StudioGTR

StudioGTR is a compact and very easy-to-use Windows-based GUI application intended to monitor the operation, generation of random numbers and random symbolic sequences, performing statistical tests and calibration of the GTR-01. Optimized user interface and simplified work with the application allow for a quick visual check of basic device performance.

You can download the software from the tech info section on the product's page: www.benev.biz/en/quantum-random-generators.html

 **NOTE:** For proper communication between StudioGTR and GTR-01, USB drivers should be installed. For more information on how to install the driver pack, please see previous section.


3.1 System requirements

Requirements	
Operating system	Windows XP*, Vista, 7, 8, 8.1, 10; 32-bit / 64-bit
CPU	1 GHz or better; 32-bit / 64-bit
RAM	1 GB or more
Screen resolution	1024 x 768 or higher
Hard disk free space	50 MB or more
USB	2.0, 3.0 or 3.1 single port
Adobe Acrobat Reader	Version 9.0 or higher

* Windows XP requires .NET Framework 2.0 or higher to be installed.

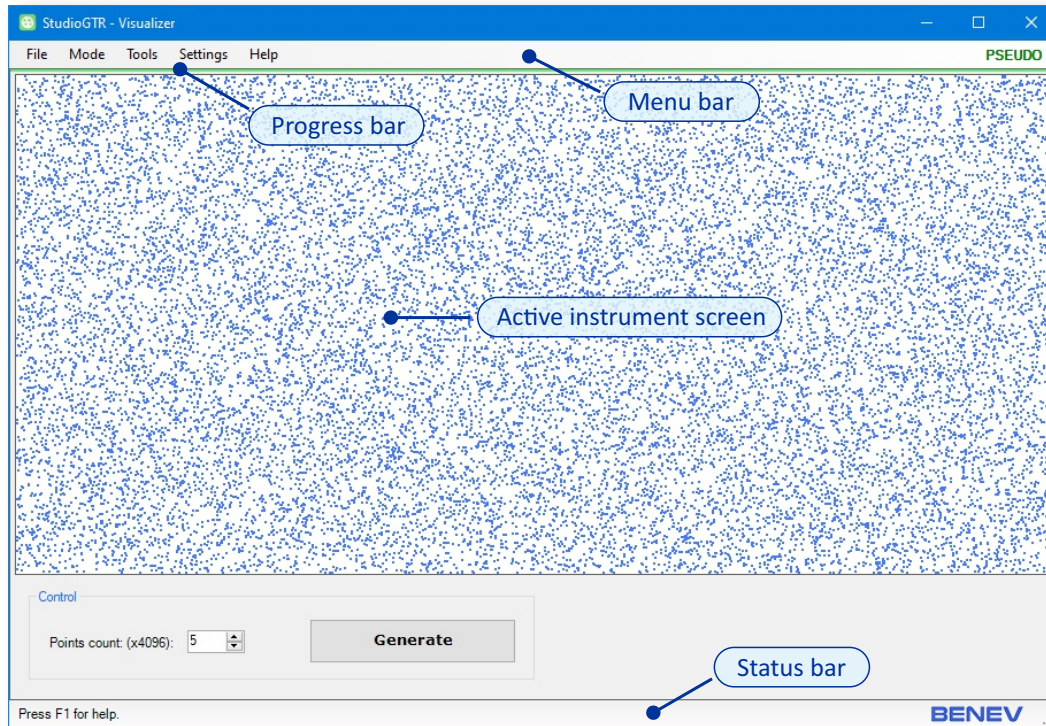
3.2 Application installation

StudioGTR does not need to be installed at all. Just unpack the StudioGTR.zip file somewhere on your system, for example, in the C:\Program Files\StudioGTR and put a shortcut to the application's .exe file on your Desktop or Start menu. That's all.

 **NOTE:** Do not change the structure and content of the unpacked zip archive.

3.3 Main screen

StudioGTR's main screen is shown on the figure below.




StudioGTR main screen.

3.4 StudioGTR basic features


StudioGTR has two modes of random number generation:

- *Pseudorandom* – uses built-in software pseudorandom number generators. To activate this mode, from the Main menu click *Mode* → *Pseudorandom*.
- *True* – uses GTR-01 as a random number generator. To activate this mode, from the Main menu click *Mode* → *True*.

 **NOTE:** The current operational mode and device status are shown on the right side of the menu-bar.

StudioGTR offers a few instruments to work with random events and numbers:

- *Visualizer* – it is used for the quick visual check of the quality of random numbers.
- *Integer Numbers* – generates random sequences of integer numbers between min and max values.
- *Float Numbers* – generates random sequences of floating-point numbers between min and max values.
- *Strings* – generates random sequences of symbols.
- *Random Noise* – generates some types of random noise.
- *Bitmaps* – generates images with a random distribution of the pixel values.
- *Lottery Numbers* – generates lucky lottery numbers.
- *Random Event Logger* – generates random numbers with sampling rate in autonomous mode and records data to the PC's hard drive.
- *Statistical Tests* – performs an automatic statistical check of the used random generator.
- *Calibration* – calibrates GTR-01 in manual or automatic mode.

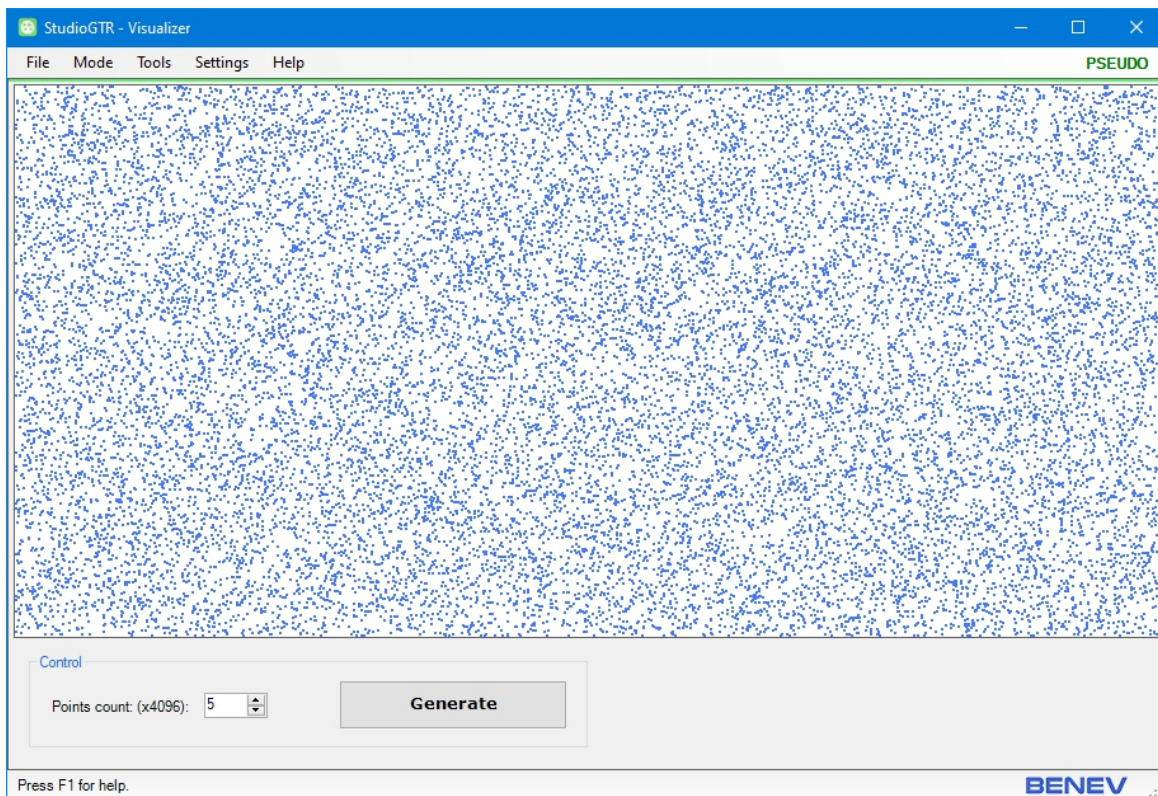
 **NOTE:** You can save the application's screen as an image using the Main menu *File* → *Save Screen...* or Ctrl+G keyboard shortcut.

3.5 Tools

3.5.1 Visualizer



To activate this mode, from the Main menu click *Tools* → *Visualizer*. It allows for a quick visual check of the generated random numbers to be performed.



StudioGTR Visualizer.

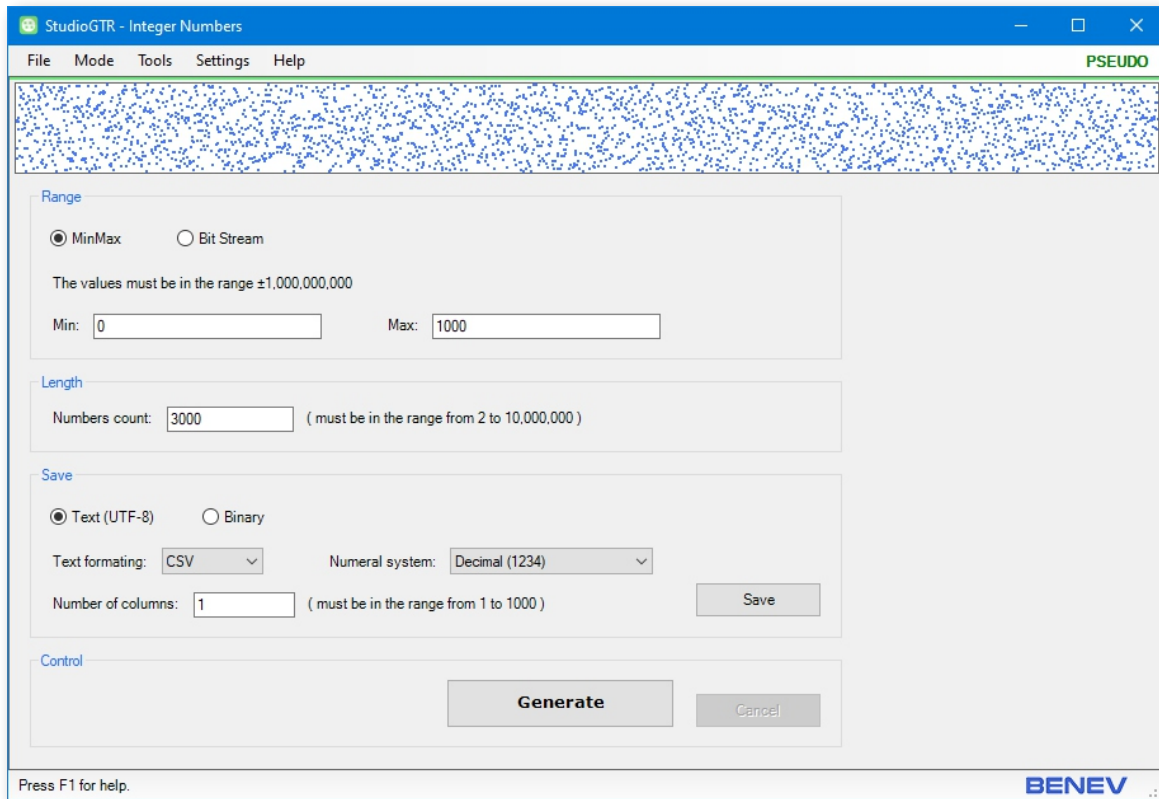


In the *Control* section, you can set the count of the generated random points.

3.5.2 Integer Numbers



To activate this mode, from the Main menu click *Tools* → *Integer Numbers*. This instrument generates integer random numbers between min and max values.



StudioGTR Integer Numbers.



In the *Range* section, you can set the min and max values. If the *Bit Stream* option is chosen, the values are in the range from -2^{31} to $+2^{31}-1$. Use this option to generate bit sequences.



In the *Length* section, you can set the count of the generated numbers.



In the *Save* section, you can save the generated random sequence to the file (text or binary). If the file is in text format, you can set the delimiter, numeral system (decimal or hexadecimal) and column number.

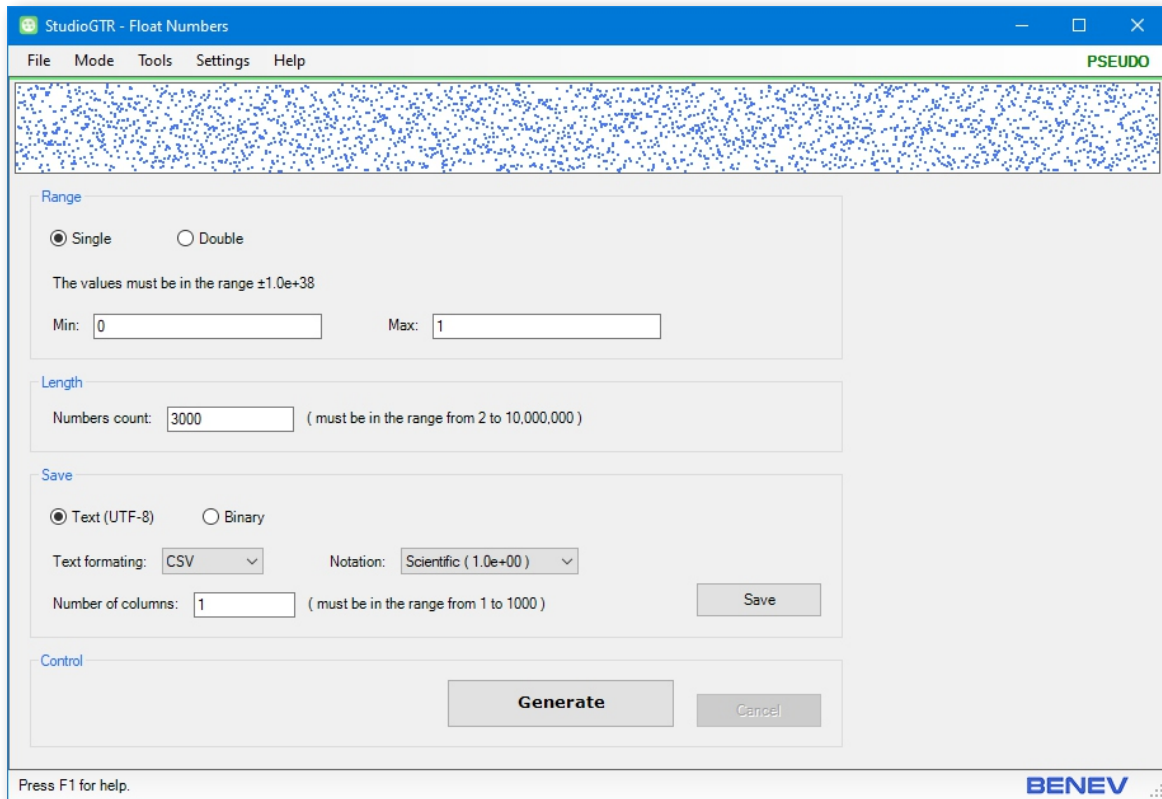


NOTE: Generated integer numbers are of type *32-bit signed integer*.

3.5.3 Float Numbers



To activate this mode, from the Main menu click *Tools* → *Float Numbers*. This instrument generates floating-point random numbers between min and max values.



StudioGTR Float Numbers.



In the *Range* section, you can set the precision (*Single* – 32-bit or *Double* – 64-bit) and the min and max values.



In the *Length* section, you can set the count of the generated numbers.



In the *Save* section, you can save the generated random sequence to the file (text or binary). If the file is in text format, you can set the delimiter, scientific notation (1.0e+00 or 1.0E+00) and column number.

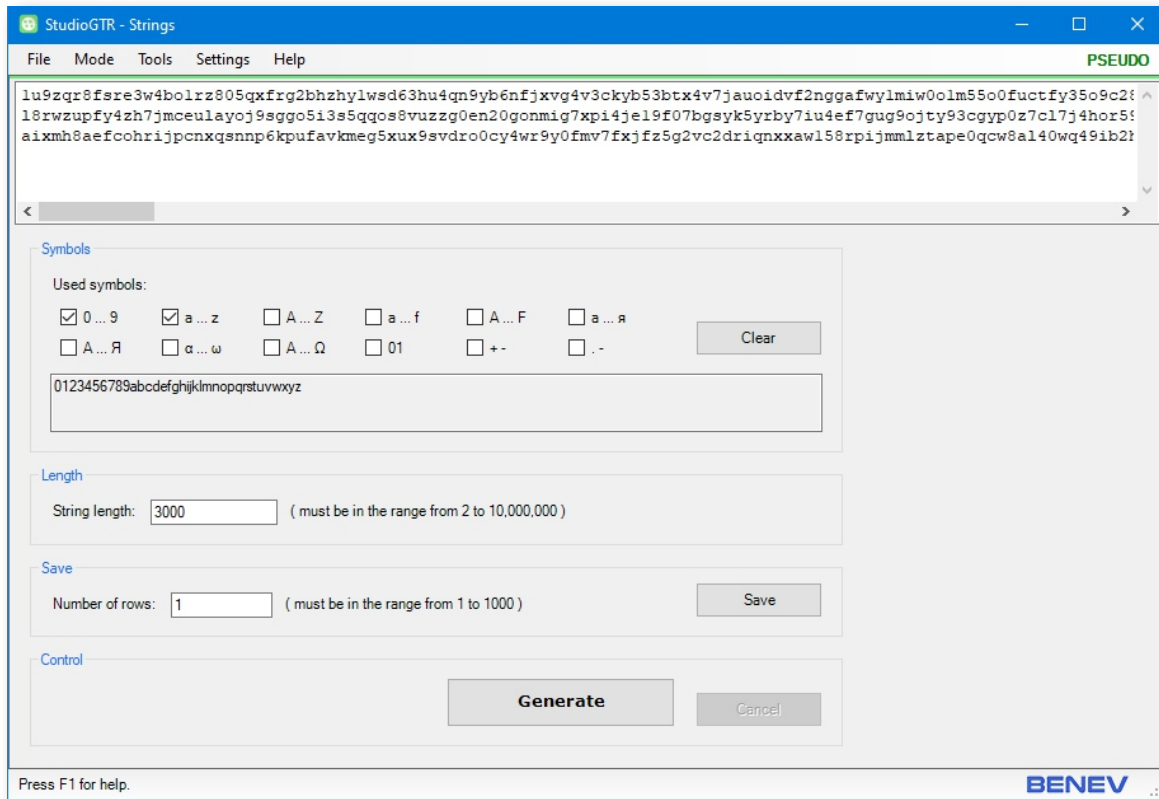


NOTE: The generated numbers use the built-in .NET formats for floating-point numbers, according to the relevant international standards.

3.5.4 Strings



To activate this mode, from the Main menu click *Tools* → *Strings*. This instrument generates character strings.



StudioGTR Strings.



In the *Symbols* section, you can choose the combination of symbol groups used to generate the string. The possible combinations are:

- 0123456789
- abcdefghijklmnopqrstuvwxyz
- ABCDEFGHIJKLMNOPQRSTUVWXYZ
- abcdef
- ABCDEF
- абвгдежзийклмнопрстуфхцчшщъьюя
- АБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЬЮЯ
- αβγδεζηθικλμνξοπρστυφχψω
- ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩ
- 01
- +-
- .-



In the *Length* section, you can set the string length.



In the *Save* section, you can set the row number.

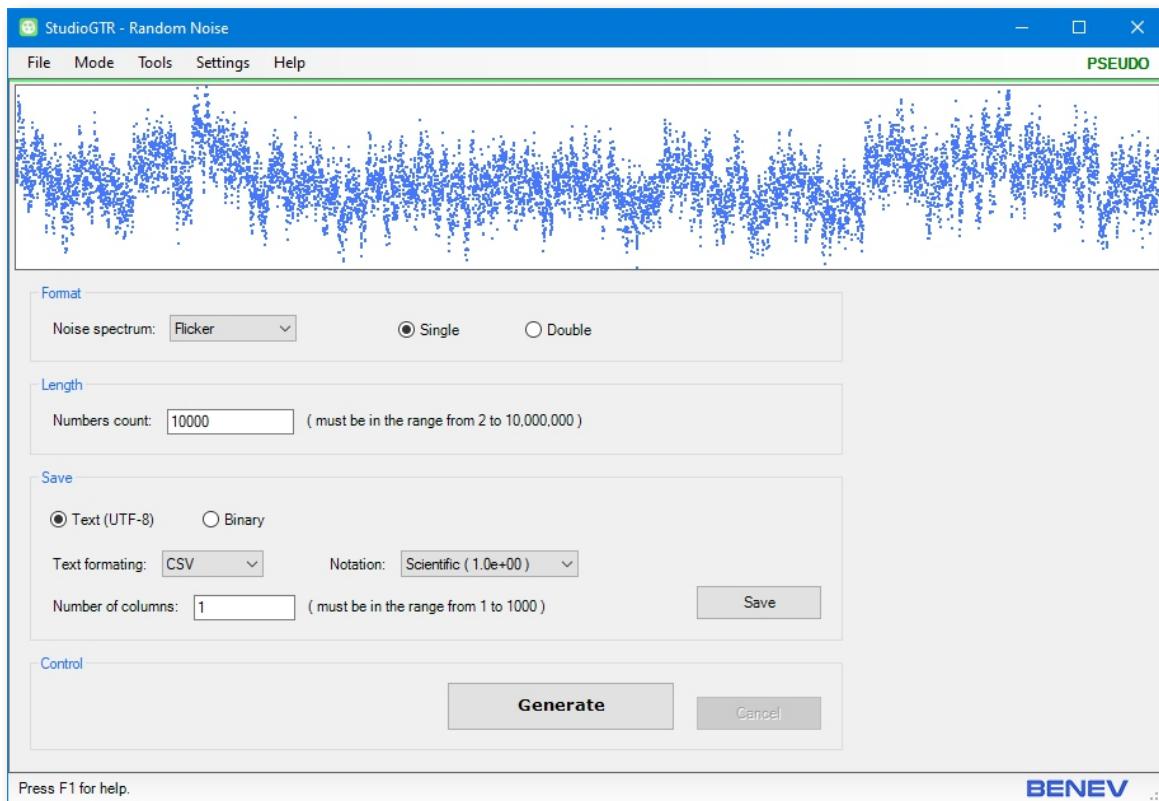


NOTE: Strings are always saved in the text format file.

3.5.5 Random Noise



To activate this mode, from the Main menu click *Tools* → *Random Noise*. This instrument generates random noise of a few types.



StudioGTR Random Noise.



In the *Format* section, you can set the noise type and the precision (*Single* – 32-bit or *Double* – 64-bit). StudioGTR generates a few types of random noise:

- *White* noise. The application generates $N(0, 1)$ numbers using *Box-Muller Transform*.
- *Flicker* noise ($1/f$ or *Pink* noise). StudioGTR uses the algorithm shown in Gardner M. “Mathematical games-white and brown music, fractal curves and one-over-f fluctuations”, *Scientific American*, 238(4):16–32, 1978.
- *Brown (Red)* noise. It is also known as *Random Walk*.



In the *Length* section, you can set the count of the generated numbers.

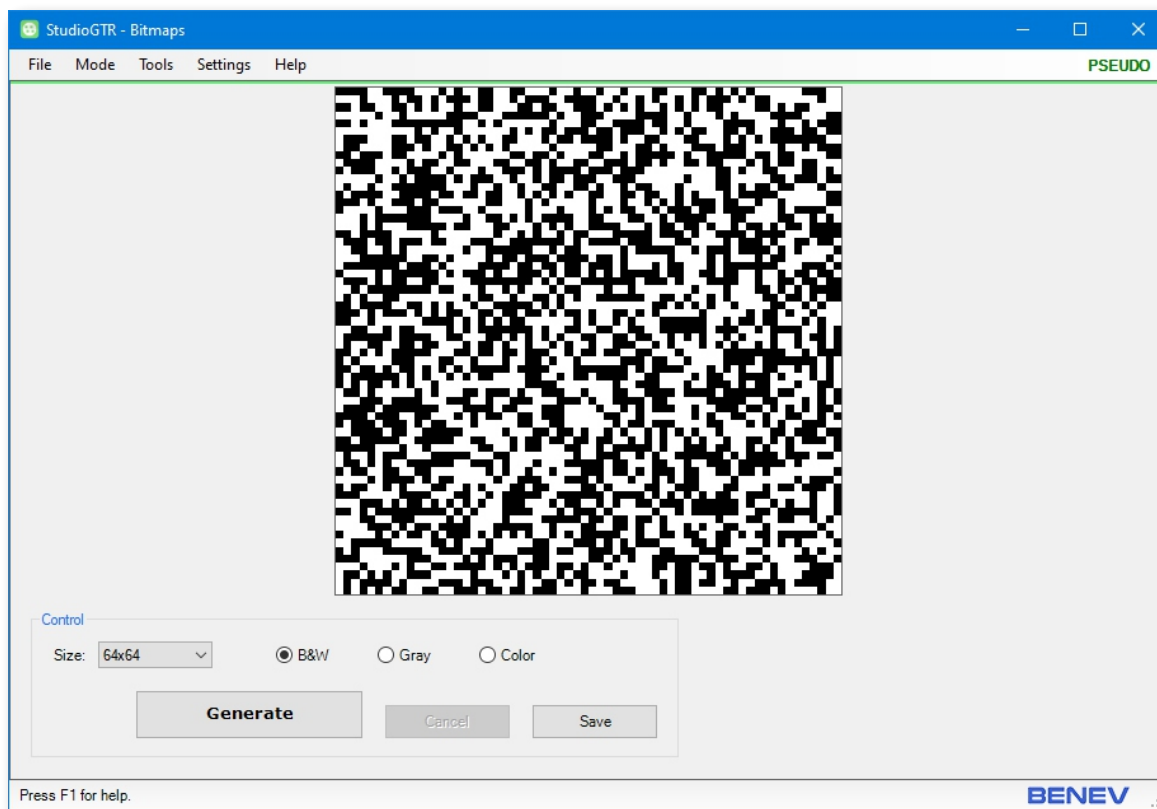


In the *Save* section, you can save the generated random sequence to the file (text or binary). If the file is in text format, you can set the delimiter, scientific notation (1.0e+00 or 1.0E+00) and column number.

3.5.6 Bitmaps



To activate this mode, from the Main menu click *Tools* → *Bitmaps*. This instrument generates square images with random distribution of the pixel values.



StudioGTR Bitmaps.



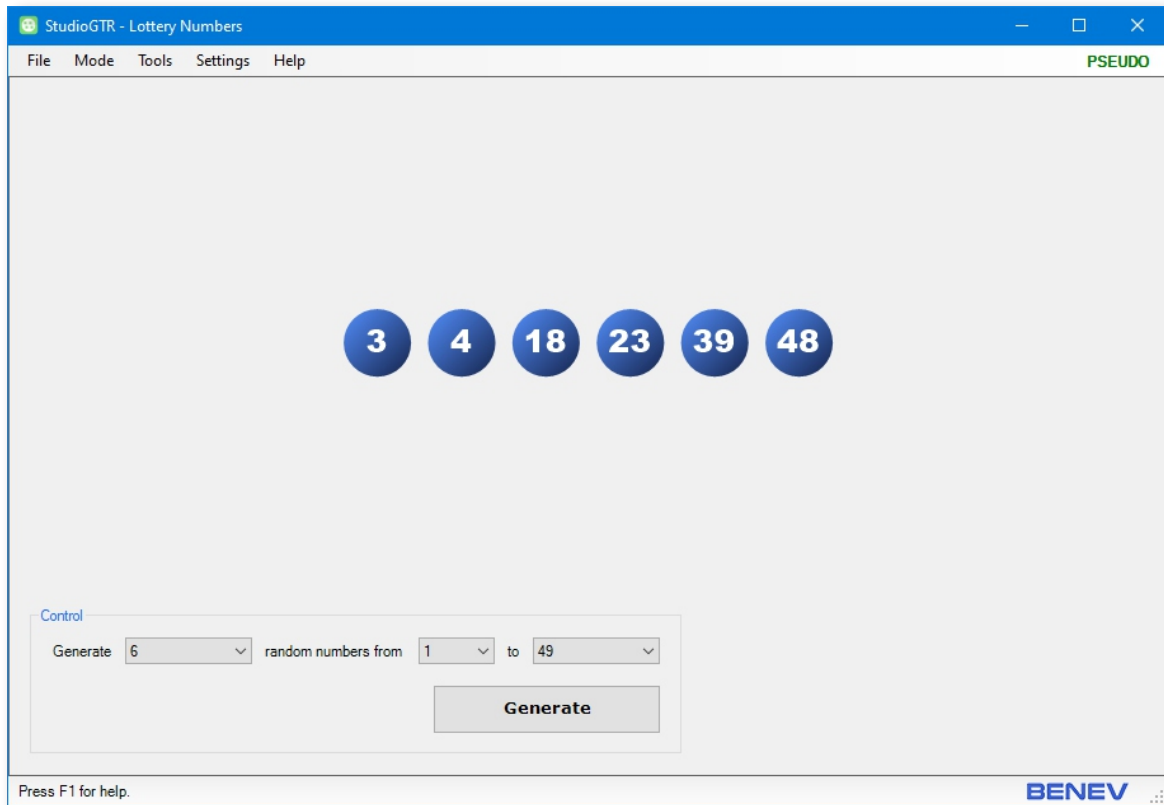
In the *Control* section, you can set the image size and type:

- *B&W*. The only pixel values are 0 (black) or 255 (white).
- *Gray*. The pixels can take monochrome values between 0 (black) and 255 (white).
- *Color*. The pixels can take RGB color values. Every color component (red, green and blue) is in the range between 0 and 255.

3.5.7 Lottery Numbers



To activate this mode, from the Main menu click *Tools* → *Lottery Numbers*. This instrument generates lucky lottery numbers.



StudioGTR Lottery Numbers.

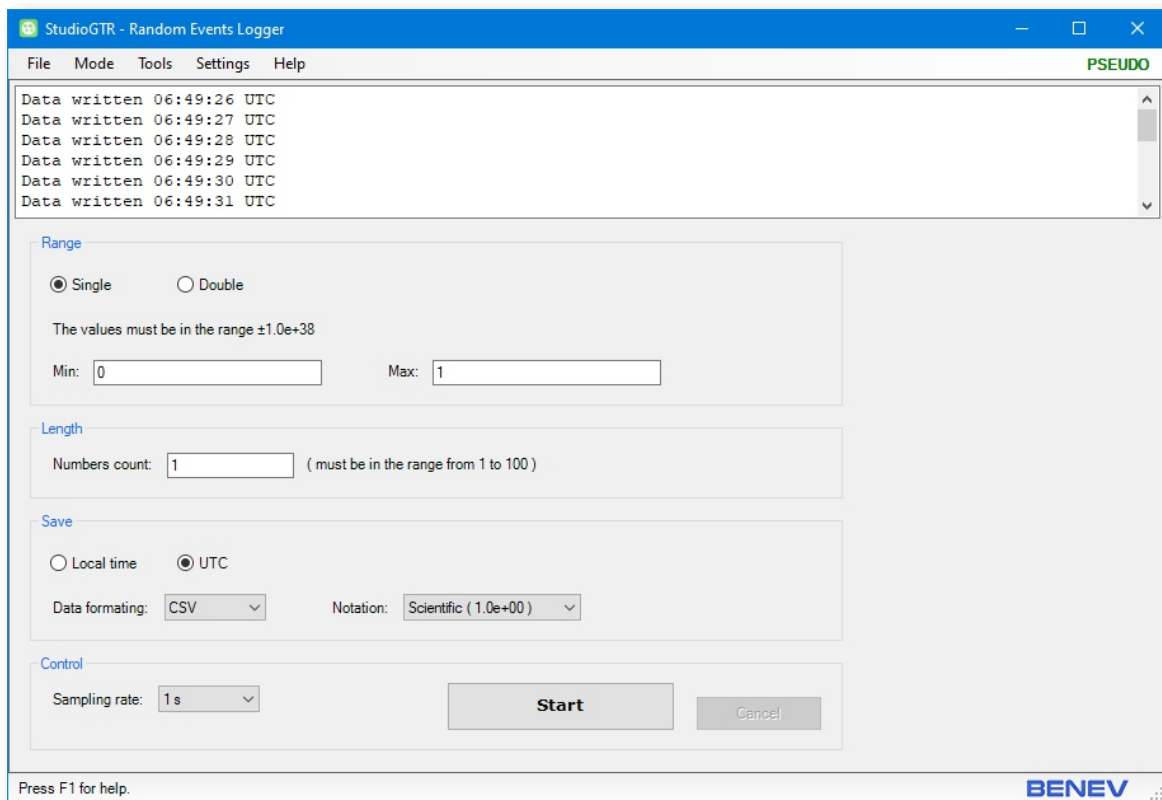


In the *Control* section, you can set the numbers count, initial value (0 or 1) and range from which to generate. For example, playing Lotto 6/49, choose 6 numbers with values from 1 to 49, playing Lotto 5/35, choose 5 numbers with values from 1 to 35, and so on.

3.5.8 Random Event Logger



To activate this mode, from the Main menu click *Tools* → *Random Event Logger*. This instrument generates random numbers with the given sampling rate in continuous mode and automatically saves the data to the Pc's hard drive. The tool only makes sense in *True* mode and is suitable for monitoring the impact of natural phenomena on the operation of the GTR-01. In the same manner, it allows the use of GTR-01 to register such phenomena.



StudioGTR Random Event Logger.



In the *Range* section, you can set the precision (*Single* – 32-bit or *Double* – 64-bit) and the min and max values. Dat are floating-point numbers.



In the *Length* section, you can set the count of the generated numbers.



In the *Save* section, you can set the recorded time type (*Local time* or *Universal Time Coordinated, UTC*), and set the delimiter and scientific notation (1.0e+00 or 1.0E+00).



NOTE: Data are always recorded in text file format. To choose the save directory, see section 3.8.

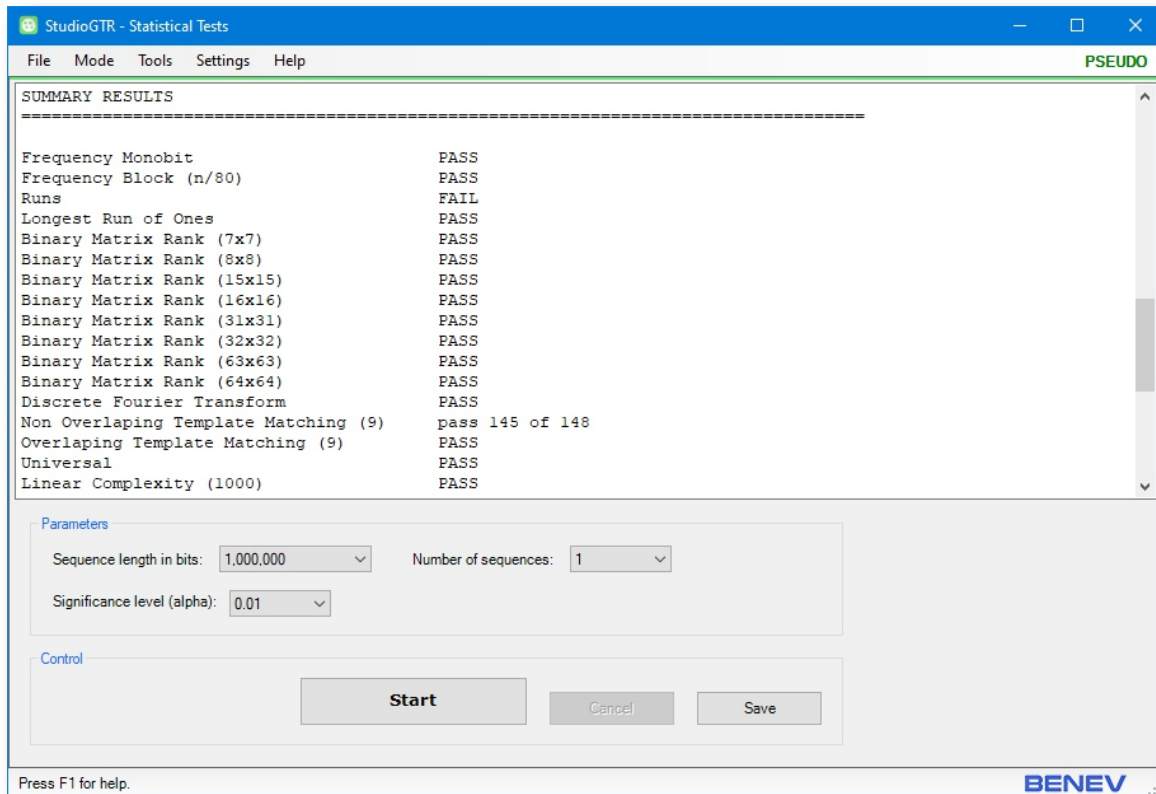


In the *Control* section, you can set the sampling rate between 1 s and 60 minutes.

3.5.9 Statistical Tests



To activate this mode, from the Main menu click *Tools* → *Statistical Tests*. This instrument checks the output quality of the currently selected random generator in the program. It implements a specially developed battery of statistical tests.



StudioGTR Statistical Tests.



In the *Parameters* section, you can set the length of the bit sequence, in bits, the number of all sequences used in the test procedure and the level of significance (alpha).




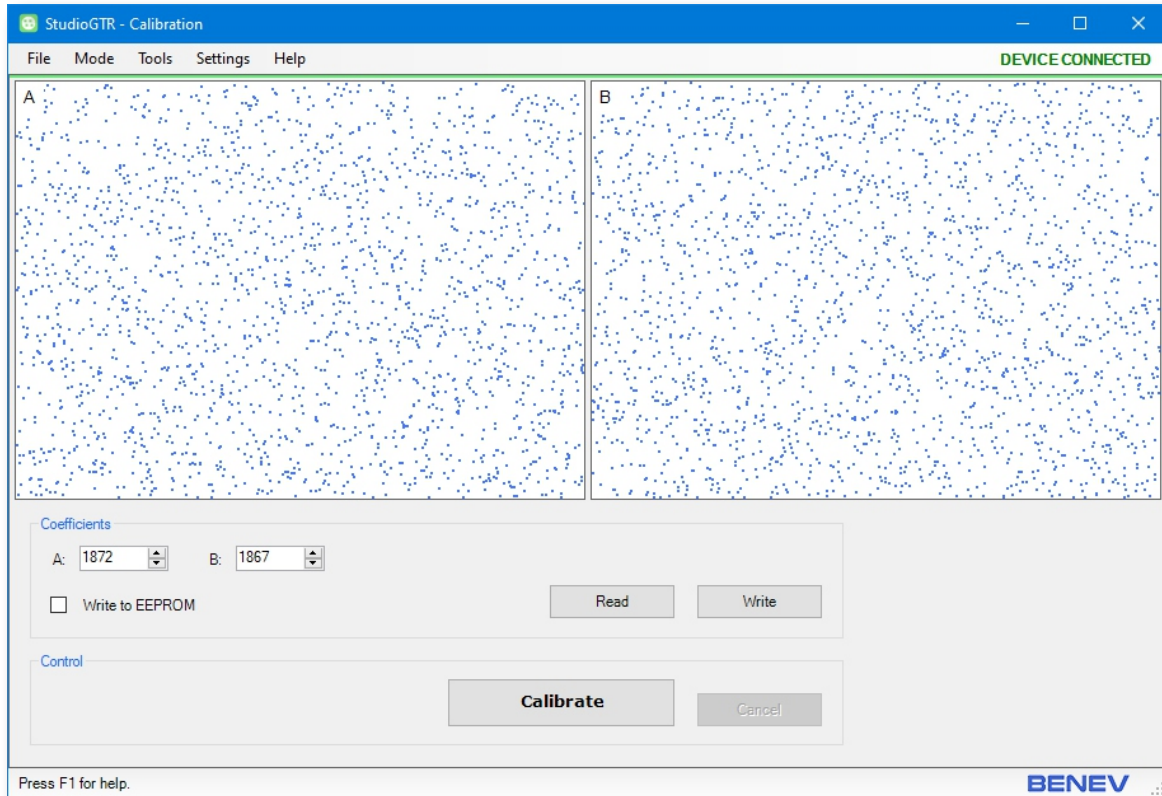
NOTE: If the sequence length and the number of sequences are in the upper limit, the test time may exceed tens of hours.




NOTE: StudioGTR uses statistical test battery published by *National Institute of Standards and Technology (NIST)*. For more info, please read “*A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Application*” on the address: <https://csrc.nist.gov/publications/detail/sp/800-22/rev-1a/final>


3.5.10 Calibration


 To activate this mode, from the Main menu click *Tools* → *Calibration*. This instrument calibrates GTR-01 in manual or automatic mode.



StudioGTR Calibration.

 In the *Coefficients* section, you can read the current values of calibration coefficients for the two generator hardware channels. You can set the new values manually and write them to the device. When the *Write to EEPROM* is set, the values will be saved to the device EEPROM.

 In the *Control* section, you can start an automatic calibration procedure and then, if you wish, write the calculated coefficient values to the GTR-01 memory.

 **NOTE:** The instrument is active in *True* mode only.

 **NOTE:** GTR-01 does not implement a *Factory Reset* function.

3.6 Settings

3.6.1 Channel



To set the active hardware channel of the GTR-01, from the Main menu click [Settings](#) → [Channel](#). The program allows the following options:

- *Channel A*. Data are generated only by channel A.
- *Channel B*. Data are generated only by channel B.
- *XOR (Channel A XOR Channel B)*. Logical bitwise XOR between two channels (default mode).

XOR plus. Same as in the previous mode but with some additional improvements. The generation speed is slightly decreased.

XOR DeBias. Same as in the default mode but with the addition of *John von Neumann's* debiasing procedure. The generation speed is decreased.

3.6.2 Transfer



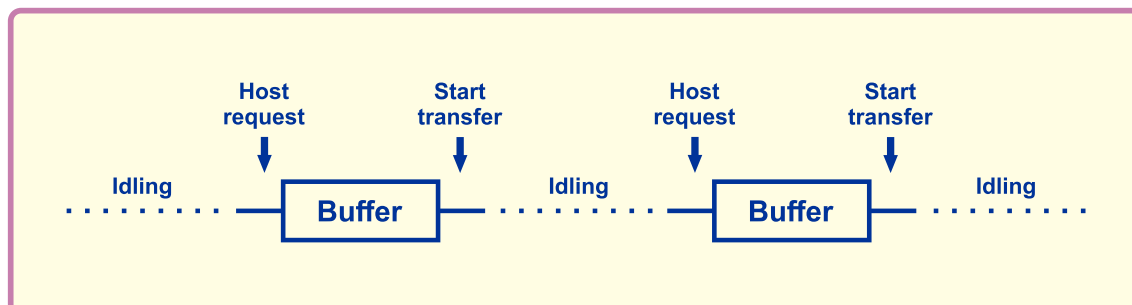
To set the transfer mode of the GTR-01, from the Main menu click [Settings](#) → [Transfer](#). The program allows the following options:

- *Synchronous* (default).
- *Asynchronous*.



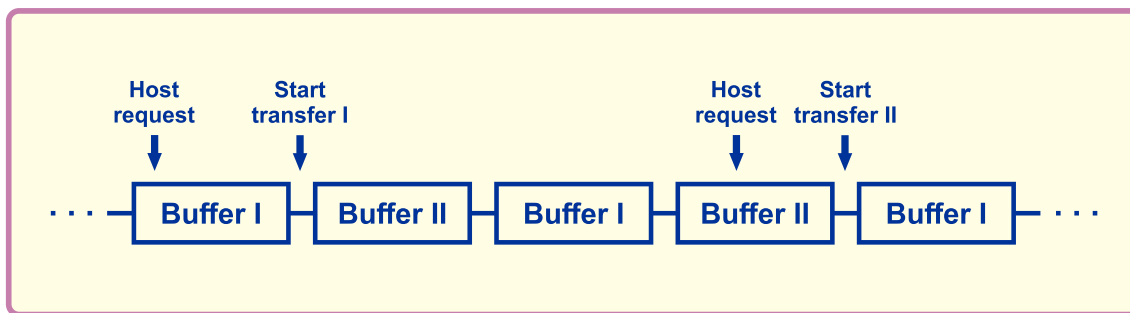
NOTE: Internally, GTR-01 implements two basic modes of data acquisition.

In the so-called *Synchronous mode*, data are collected in a single buffer whenever a data request from the host (PC) has been made. In such a way, the device (GTR-01) is idling until a new request is received and data packets are synchronized with the host operation, providing good predictability and time precision. This mode is suitable in situations where the exact moments of data acquisition should be controlled by the host.



Synchronous mode of operation.

On the other side, the so-called *Asynchronous mode* implements internally the *Ping-pong scheme* where data are continuously collected in the two separate data buffers and independently from the host requests. After one buffer has been filled with data, acquisition continues in the other one without any interruption and so on. The two buffers are constantly swapping one after another, hence the name ping-pong. When the host request arrives, the transfer of the last ready buffer filled with fresh data can start immediately, and in such a way, no time is lost in waiting for the buffer to be filled. This mode is suitable when the main goal is a maximum speed of operation with the highest possible transfer rate.



Asynchronous mode of operation.

3.6.3 Pseudorandom Generator



To set the type of the pseudorandom generator used in StudioGTR, from the Main menu click *Settings* → *Pseudorandom Generator*. The program allows the following options:

- *.NET* (default). *.NET pseudorandom number generator*.
- *.NET Crypto*. *.NET cryptographically secure pseudorandom number generator*.
- *LCG* (*Linear congruential generator*).
- *MWC* (*Multiply-with-carry*). The generator is invented by *George Marsaglia*.
- *Xorshift*. The generator is invented by *George Marsaglia*.
- *Combo* (*LCG+MWC+Xorshift*).

3.7 Multilingual support

StudioGTR is an application designed with built-in multilingual support .



To change the current application language:

- From the Main menu click on *Settings* → *Language* to select the language you prefer.
- In the popup dialog box, confirm your selection and StudioGTR will automatically close and restart with the new language settings.



NOTE: When the application is starting for the first time on your computer, the default language is *English*.



NOTE: The application help file launches in the language version corresponding to the current application language.

3.8 Application preferences

StudioGTR allows for customization through setting user's preferences .

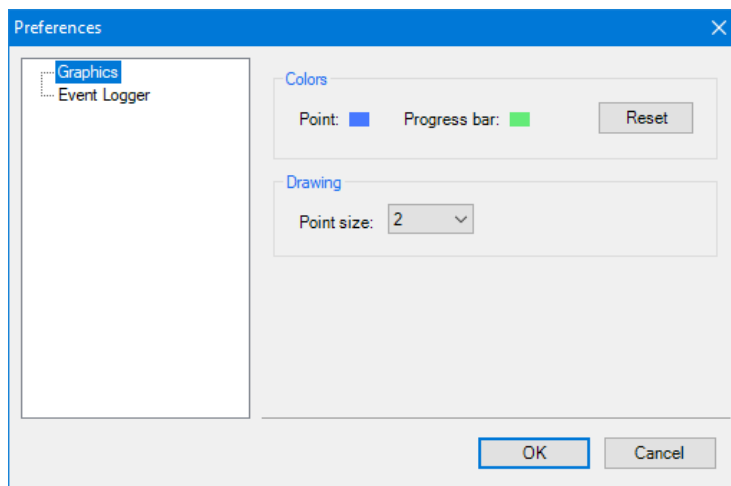


To launch the Preferences box, from the Main menu click on *Settings* → *Preferences...*

- In the *Graphics* page you can choose the point and progress bar colors and set the point size. Click on the Reset button if you want to set back colors to their default values.

- In the *Event Logger* page you can choose the directory path for automatic data recording.

When you are ready click OK button. The preferences will take place immediately. On the next launch StudioGTR will start with the new settings.





Preferences dialog box.

4. Low level API

GTR-01 is specially designed as a high-quality, high-speed source of true random numbers. It can be used in areas requiring a high degree of randomness and unpredictability. And if the StudioGTR application allows for some basic observations and calibration procedures to be performed, the complete set of all capabilities, embedded in the GTR-01, can be achieved only through the power of the low-level device communication. This approach requires some basic programming skills. See Appendix A and Appendix B of this guide for more examples, using Microsoft C# programming language.

GTR-01 utilizes an FTDI USB interface chip and needs an FTDI driver pack to be installed on the PC. Please read the driver documentation and application notes on the driver vendor's site.

 **NOTE:** Communicating with the GTR-01 requires USB drivers to be installed first. For more information on how to install the driver pack, please see section 3.5.

 **NOTE:** For maximum transfer speed, use asynchronous IO commands. Otherwise, use synchronous ones.

4.1 GTR-01 IO commands

Description of the low level GTR-01 IO commands is given below.

 **IMPORTANT:** You must always write 64 bytes to the GTR-01 and read back 4096 bytes!

IO Command	Value
GET_RND_CHANNEL_A	0x01
GET_RND_CHANNEL_A_CALIBRATION	0x02
GET_RND_CHANNEL_B	0x11
GET_RND_CHANNEL_B_CALIBRATION	0x12
GET_RND_NORMAL	0x21
GET_RND_NORMAL_PLUS	0x22
GET_RND_NORMAL_DEBIAS	0x23
GET_RND_CHANNEL_A_ASYNC	0x31
GET_RND_CHANNEL_A_CALIBRATION_ASYNC	0x32
GET_RND_CHANNEL_B_ASYNC	0x41
GET_RND_CHANNEL_B_CALIBRATION_ASYNC	0x42
GET_RND_NORMAL_ASYNC	0x51
GET_RND_NORMAL_PLUS_ASYNC	0x52
GET_RND_NORMAL_DEBIAS_ASYNC	0x53
SET_CALIBRATION	0x80
GET_CALIBRATION	0x81
GET_DEVICE_ID	0x82
EEPROM_WRITE*	0x55

* The command should be written to the second byte. See the explanation below.

4.1.1 GET_RND_CHANNEL_A

Write:

Byte0	0x01
Byte1 to Byte63	No matter

Read:

Byte0 to Byte4095	Random bytes from channel A, synchronous transfer
-------------------	---

4.1.2 GET_RND_CHANNEL_A_CALIBRATION

Write:

Byte0	0x02
Byte1 to Byte63	No matter

Read:

Byte0 to Byte4095	Random bytes from channel A, synchronous transfer. The bytes returned are “as is” without any software improvements. Use this command when calibrating channel A.
-------------------	---

4.1.3 GET_RND_CHANNEL_B

Write:

Byte0	0x11
Byte1 to Byte63	No matter

Read:

Byte0 to Byte4095	Random bytes from channel B, synchronous transfer
-------------------	---

4.1.4 GET_RND_CHANNEL_B_CALIBRATION

Write:

Byte0	0x12
Byte1 to Byte63	No matter

Read:

Byte0 to Byte4095	Random bytes from channel B, synchronous transfer. The bytes returned are “as is” without any software improvements. Use this command when calibrating channel B.
-------------------	---

4.1.5 GET_RND_NORMAL

Write:

Byte0	0x21
Byte1 to Byte63	No matter

Read:

Byte0 to Byte4095	Random bytes from channel A XOR channel B, synchronous transfer
-------------------	---

4.1.6 GET_RND_NORMAL_PLUS

Write:

Byte0	0x22
Byte1 to Byte63	No matter

Read:

Byte0 to Byte4095	Random bytes from channel A XOR channel B, synchronous transfer. Identical to GET_RND_NORMAL with some software improvements of the output data quality.
-------------------	--

4.1.7 GET_RND_NORMAL_DEBIAS

Write:

Byte0	0x23
Byte1 to Byte63	No matter

Read:

Byte0 to Byte4095	Random bytes from channel A XOR channel B, synchronous transfer. Implements John von Neumann's debiasing algorithm.
-------------------	---

4.1.8 GET_RND_CHANNEL_A_ASYNC

Write:

Byte0	0x31
Byte1 to Byte63	No matter

Read:

Byte0 to Byte4095	Random bytes from channel A, asynchronous transfer
-------------------	--

4.1.9 GET_RND_CHANNEL_A_CALIBRATION_ASYNC

Write:

Byte0	0x32
Byte1 to Byte63	No matter

Read:

Byte0 to Byte4095	Random bytes from channel A, asynchronous transfer. The bytes returned are "as is" without any software improvements. Use this command when calibrating channel A.
-------------------	--

4.1.10 GET_RND_CHANNEL_B_ASYNC

Write:

Byte0	0x41
Byte1 to Byte63	No matter

Read:

Byte0 to Byte4095	Random bytes from channel B, asynchronous transfer
-------------------	--

4.1.11 GET_RND_CHANNEL_B_CALIBRATION_ASYNC

Write:

Byte0	0x42
Byte1 to Byte63	No matter

Read:

Byte0 to Byte4095	Random bytes from channel B, asynchronous transfer. The bytes returned are “as is” without any software improvements. Use this command when calibrating channel B.
-------------------	--

4.1.12 GET_RND_NORMAL_ASYNC

Write:

Byte0	0x51
Byte1 to Byte63	No matter

Read:

Byte0 to Byte4095	Random bytes from channel A XOR channel B, asynchronous transfer
-------------------	--

4.1.13 GET_RND_NORMAL_PLUS_ASYNC

Write:

Byte0	0x52
Byte1 to Byte63	No matter

Read:

Byte0 to Byte4095	Random bytes from channel A XOR channel B, asynchronous transfer. Identical to GET_RND_NORMAL_ASYNC with some software improvements of the output data quality.
-------------------	---

4.1.14 GET_RND_NORMAL_DEBIAS_ASYNC

Write:

Byte0	0x53
Byte1 to Byte63	No matter

Read:

Byte0 to Byte4095	Random bytes from channel A XOR channel B, asynchronous transfer. Implements John von Neumann's debiasing algorithm.
-------------------	--

4.1.15 SET_CALIBRATION

Write:

Byte0	0x80
Byte1	0x55 - Write to EEPROM; Otherwise - Don't write to EEPROM
Byte2	Reserved
Byte3	Reserved
Byte4	Channel A calibration value, 12-bit unsigned, byte0
Byte5	Channel A calibration value, 12-bit unsigned, byte1
Byte6	Channel B calibration value, 12-bit unsigned, byte0
Byte7	Channel B calibration value, 12-bit unsigned, byte1
Byte8 to Byte63	No matter

Read:

Byte0	EEPROM write status, 0 means OK
Byte1 to Byte4095	No meaning

4.1.16 GET_CALIBRATION

Write:

Byte0	0x81
Byte1 to Byte63	No matter

Read:

Byte0	Channel A calibration value, 12-bit unsigned, byte0
Byte1	Channel A calibration value, 12-bit unsigned, byte1
Byte2	Channel B calibration value, 12-bit unsigned, byte0
Byte3	Channel B calibration value, 12-bit unsigned, byte1
Byte4 to Byte4095	No meaning

4.1.17 GET_DEVICE_ID

Write:

Byte0	0x82
Byte1 to Byte63	No matter

Read:

Byte0	Device ID, 64-bit unsigned integer, byte0
Byte1	Device ID, 64-bit unsigned integer, byte1
Byte2	Device ID, 64-bit unsigned integer, byte2
Byte3	Device ID, 64-bit unsigned integer, byte3
Byte4	Device ID, 64-bit unsigned integer, byte4
Byte5	Device ID, 64-bit unsigned integer, byte5
Byte6	Device ID, 64-bit unsigned integer, byte6
Byte7	Device ID, 64-bit unsigned integer, byte7
Byte8 to Byte4095	No meaning

Appendix A: Statistical tests

The randomness and unpredictability of the generated output data stream numbers have been validated using the following statistical test batteries:

Test	Website
NIST STS	https://csrc.nist.gov/projects/random-bit-generation
TestU01	http://simul.iro.umontreal.ca/testu01/tu01.html
Diehard	https://web.archive.org/web/20160125103112/http://stat.fsu.edu/pub/diehard/



NOTE: More information about the test procedure you can find here:
<https://benev.biz/en/statistical-testing-random-generators.html>

Appendix B: FTDI IO methods wrapper in C#

This is an example of using C# programming language and P/Invoke technology to wrap some basic IO methods, originally written in C and described in the *D2XX Programmer's Guide* at <https://ftdichip.com>. The code shown below is located in the GTR_IO_wrap.cs file found in the StudioGTR.zip archive.

```
using System;
using System.Text;
using System.Runtime.InteropServices;

namespace GTR
{
    // FT_STATUS enumeration.
    public enum FT_STATUS : uint
    {
        FT_OK = 0,
        FT_INVALID_HANDLE,
        FT_DEVICE_NOT_FOUND,
        FT_DEVICE_NOT_OPENED,
        FT_IO_ERROR,
        FT_INSUFFICIENT_RESOURCES,
        FT_INVALID_PARAMETER,
        FT_INVALID_BAUD_RATE,
        FT_DEVICE_NOT_OPENED_FOR_ERASE,
        FT_DEVICE_NOT_OPENED_FOR_WRITE,
        FT_FAILED_TO_WRITE_DEVICE,
        FT_EEPROM_READ_FAILED,
        FT_EEPROM_WRITE_FAILED,
        FT_EEPROM_ERASE_FAILED,
        FT_EEPROM_NOT_PRESENT,
        FT_EEPROM_NOT_PROGRAMMED,
        FT_INVALID_ARGS,
        FT_NOT_SUPPORTED,
        FT_OTHER_ERROR
    }

    // FT_DEVICE_LIST_INFO_NODE struct.
    [StructLayout(LayoutKind.Sequential)]
    public struct FT_DEVICE_LIST_INFO_NODE
    {
        public uint Flags;
        public uint Type;
        public uint ID;
        public uint LocId;
        [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 16)]
        public string SerialNumber;
        [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 64)]
        public string Description;
        public IntPtr ftHandle;
    }
}
```

```

// FTDI class.
public class FTDI
{
    // Flags.
    public const uint FT_LIST_NUMBER_ONLY = 0x80000000u;
    public const uint FT_LIST_BY_INDEX = 0x40000000u;
    public const uint FT_LIST_ALL = 0x20000000u;

    public const uint FT_OPEN_BY_SERIAL_NUMBER = 1u;
    public const uint FT_OPEN_BY_DESCRIPTION = 2u;
    public const uint FT_OPEN_BY_LOCATION = 4u;

    // Word Lengths.
    public const byte FT_BITS_8 = (byte)8;
    public const byte FT_BITS_7 = (byte)7;

    // Stop Bits.
    public const byte FT_STOP_BITS_1 = (byte)0;
    public const byte FT_STOP_BITS_2 = (byte)2;

    // Parity.
    public const byte FT_PARITY_NONE = (byte)0;
    public const byte FT_PARITY_ODD = (byte)1;
    public const byte FT_PARITY_EVEN = (byte)2;
    public const byte FT_PARITY_MARK = (byte)3;
    public const byte FT_PARITY_SPACE = (byte)4;

    // Flow Control.
    public const ushort FT_FLOW_NONE = (ushort)0x0000;
    public const ushort FT_FLOW_RTS_CTS = (ushort)0x0100;
    public const ushort FT_FLOW_DTR_DSR = (ushort)0x0200;
    public const ushort FT_FLOW_XON_XOFF = (ushort)0x0400;

    // Purge rx and tx buffers.
    public const uint FT_PURGE_RX = 1;
    public const uint FT_PURGE_TX = 2;

    // Classic Interface Methods.
    [DllImport("FTD2XX.dll")]
    public static extern FT_STATUS FT_CreateDeviceInfoList(ref uint lpdwNumDevs);

    [DllImport("FTD2XX.dll")]
    public static extern FT_STATUS FT_GetDeviceInfoList(
        [In, Out] FT_DEVICE_LIST_INFO_NODE[] pDest,
        ref uint lpdwNumDevs);

    [DllImport("FTD2XX.dll")]
    public static extern FT_STATUS FT_ListDevices(
        ref uint pvArg1,
        uint pvArg2,
        uint dwFlags);

    [DllImport("FTD2XX.dll")]
    public static extern FT_STATUS FT_Open(
        int iDevice,
        ref IntPtr ftHandle);

    [DllImport("FTD2XX.dll")]
    public static extern FT_STATUS FT_Close(IntPtr ftHandle);

    [DllImport("FTD2XX.dll")]
    public static extern FT_STATUS FT_Read(
        IntPtr ftHandle,
        [In, Out] byte[] lpBuffer,
        uint dwBytesToRead,
        ref uint lpdwBytesReturned);

    [DllImport("FTD2XX.dll")]
    public static extern FT_STATUS FT_Write(
        IntPtr ftHandle,
        [In, Out] byte[] lpBuffer,
        uint dwBytesToWrite,
        ref uint lpdwBytesWritten);

    [DllImport("FTD2XX.dll")]
    public static extern FT_STATUS FT_SetBaudRate(
        IntPtr ftHandle,
        uint dwBaudRate);

    [DllImport("FTD2XX.dll")]
    public static extern FT_STATUS FT_SetDivisor(
        IntPtr ftHandle,
        ushort usDivisor);

```

```
[DllImport("FTD2XX.dll")]
public static extern FT_STATUS FT_SetDataCharacteristics(IntPtr ftHandle,
    byte uWordLength,
    byte uStopBits,
    byte uParity);

[DllImport("FTD2XX.dll")]
public static extern FT_STATUS FT_SetTimeouts(IntPtr ftHandle,
    uint dwReadTimeout,
    uint dwWriteTimeout);

[DllImport("FTD2XX.dll")]
public static extern FT_STATUS FT_SetFlowControl(IntPtr ftHandle,
    ushort usFlowControl,
    byte uXon,
    byte uXoff);

[DllImport("FTD2XX.dll")]
public static extern FT_STATUS FT_GetDeviceInfo(IntPtr ftHandle,
    ref uint pftType,
    ref uint lpdwID,
    StringBuilder pcSerialNumber,
    StringBuilder pcDescription,
    int pvDummy);

[DllImport("FTD2XX.dll")]
public static extern FT_STATUS FT_GetDriverVersion(IntPtr ftHandle,
    ref uint lpdwDriverVersion);

[DllImport("FTD2XX.dll")]
public static extern FT_STATUS FT_GetLibraryVersion(ref uint lpdwDLLVersion);

[DllImport("FTD2XX.dll")]
public static extern FT_STATUS FT_Purge(IntPtr ftHandle,
    uint dwMask);

[DllImport("FTD2XX.dll")]
public static extern FT_STATUS FT_ResetDevice(IntPtr ftHandle);

[DllImport("FTD2XX.dll")]
public static extern FT_STATUS FT_ResetPort(IntPtr ftHandle);

[DllImport("FTD2XX.dll")]
public static extern FT_STATUS FT_CyclePort(IntPtr ftHandle);

[DllImport("FTD2XX.dll")]
public static extern FT_STATUS FT_Rescan();

[DllImport("FTD2XX.dll")]
public static extern FT_STATUS FT_Reload(ushort wVID,
    ushort wPID);

// Extended API Functions.
[DllImport("FTD2XX.dll")]
public static extern FT_STATUS FT_SetLatencyTimer(IntPtr ftHandle,
    byte ucTimer);

[DllImport("FTD2XX.dll")]
public static extern FT_STATUS FT_GetLatencyTimer(IntPtr ftHandle,
    ref byte pucTimer);

[DllImport("FTD2XX.dll")]
public static extern FT_STATUS FT_SetBitMode(IntPtr ftHandle,
    byte ucMask,
    byte ucMode);

[DllImport("FTD2XX.dll")]
public static extern FT_STATUS FT_GetBitMode(IntPtr ftHandle,
    ref byte pucMode);

[DllImport("FTD2XX.dll")]
public static extern FT_STATUS FT_SetUSBParameters(IntPtr ftHandle,
    uint dwInTransferSize,
    uint dwOutTransferSize);
}
}
```

Appendix C: GTR-01 IO example

This is an example describing GTR-01 basic IO communication. The code is written in C# and is located in the class `GTR_IO` in `GTR_IO_ex.cs` file. The example uses an interface wrapper in the `GTR_IO_wrap.cs` file. Both source files are located in the `StudioGTR.zip` archive.

```
using System;

namespace GTR
{
    // GTR_IO class.
    public class GTR_IO
    {
        // Constants.
        private const ushort VID = 0x0403;
        private const ushort PID = 0x6015;           // FTDI's original PID and VID.

        private const uint BYTES_TO_WRITE           = 64u;
        private const uint BYTES_TO_READ           = 4096u;

        private const byte EEPROM_WRITE_CONST      = (byte)0x55;

        public const byte GET_RND_CHANNEL_A_REQUEST = (byte)0x01;
        public const byte GET_RND_CHANNEL_A_CALIBRATION_REQUEST = (byte)0x02;
        public const byte GET_RND_CHANNEL_B_REQUEST = (byte)0x11;
        public const byte GET_RND_CHANNEL_B_CALIBRATION_REQUEST = (byte)0x12;
        public const byte GET_RND_NORMAL_REQUEST = (byte)0x21;
        public const byte GET_RND_NORMAL_PLUS_REQUEST = (byte)0x22;
        public const byte GET_RND_NORMAL_DEBIAS_REQUEST = (byte)0x23;
        public const byte GET_RND_CHANNEL_A_ASYNC_REQUEST = (byte)0x31;
        public const byte GET_RND_CHANNEL_A_CALIBRATION_ASYNC_REQUEST = (byte)0x32;
        public const byte GET_RND_CHANNEL_B_ASYNC_REQUEST = (byte)0x41;
        public const byte GET_RND_CHANNEL_B_CALIBRATION_ASYNC_REQUEST = (byte)0x42;
        public const byte GET_RND_NORMAL_ASYNC_REQUEST = (byte)0x51;
        public const byte GET_RND_NORMAL_PLUS_ASYNC_REQUEST = (byte)0x52;
        public const byte GET_RND_NORMAL_DEBIAS_ASYNC_REQUEST = (byte)0x53;

        private const byte SET_CALIBRATION_REQUEST = (byte)0x80;
        private const byte GET_CALIBRATION_REQUEST = (byte)0x81;
        private const byte GET_DEVICE_ID_REQUEST = (byte)0x82;

        // Constructor.
        public GTR_IO()
        {
        }
    }
}
```

```
// Public methods.
public void GTR_Basic_IO()
{
    IntPtr handle = IntPtr.Zero;

    try
    {
        FT_STATUS status = FT_STATUS.FT_OK;
        uint numDevs = 0u;

        // Check for GTR device.
        status = FTDI.FT_CreateDeviceInfoList(ref numDevs);

        FT_DEVICE_LIST_INFO_NODE[] vDevInfo =
            new FT_DEVICE_LIST_INFO_NODE[numDevs];
        status = FTDI.FT_GetDeviceInfoList(vDevInfo, ref numDevs);

        string vidpid = String.Format("{0:X4}{1:X4}", GTR_IO.VID, GTR_IO.PID);

        if (status == FT_STATUS.FT_OK &&
            String.Format("{0:X8}", vDevInfo[0].ID) == vidpid)
        {
            // GTR is connected.
        }
        else
        {
            // There is something wrong or device is not connected.
        }

        // Open GTR device.
        if (FTDI.FT_Open(0, ref handle) != FT_STATUS.FT_OK)
        {
            throw new Exception("Device not found!");
        }

        // Initialize GTR device.
        FTDI.FT_SetBaudRate(handle, 3000000u);
        FTDI.FT_SetDataCharacteristics( handle,
                                        FTDI.FT_BITS_8,
                                        FTDI.FT_STOP_BITS_1,
                                        FTDI.FT_PARITY_NONE);
        FTDI.FT_SetFlowControl(handle, FTDI.FT_FLOW_RTS_CTS, 0, 0);
        FTDI.FT_SetTimeouts(handle, 3000u, 1000u);
        FTDI.FT_Purge(handle, FTDI.FT_PURGE_RX | FTDI.FT_PURGE_TX);

        // IO communication.
        uint bytesToWrite = GTR_IO.BYTES_TO_WRITE;
        uint bytesToRead = GTR_IO.BYTES_TO_READ;
        uint bytesWritten = 0u;
        uint bytesReturned = 0u;
        byte[] bufferWrite = new byte[bytesToWrite];
        byte[] bufferRead = new byte[bytesToRead];

        bufferWrite[0] = GTR_IO.GET_RND_NORMAL_REQUEST;

        status = FTDI.FT_Write(handle,
                               bufferWrite,
                               bytesToWrite,
                               ref bytesWritten);
        status = FTDI.FT_Read(handle,
                              bufferRead,
                              bytesToRead,
                              ref bytesReturned);
        if (status != FT_STATUS.FT_OK)
        {
            throw new Exception("USB communication problem!");
        }
    }
}
```

```
        // Displaying the result.
        for (int i = 0; i < bufferRead.Length; i++)
        {
            Console.Write("{0:x2}", bufferRead[i]);
        }
        Console.WriteLine();
    }
    catch (Exception exc)
    {
        // If something goes wrong, show the explanation here.
        Console.WriteLine(exc.Message);
    }
    finally
    {
        // When the IO is completed, close GTR device.
        FTDI.FT_Close(handle);
    }
}
}
```

Index

Asynchronous mode	23	Multiply-with-carry	23
Avalanche noise	1, 3	NIST	20
Box-Muller transform	16	N-region	3
Breakdown voltage	3	Ping-pong	23
Brown noise	16	P-n junction	3
Calibration values	21	P-region	3
Comparator	4	Quantum noise	3
Cryptography	1, 23	Random walk	16
Debiasing	22, 29, 31	Red noise	16
Depletion region	3	Reverse biased voltage	3
Digital-to-analog converter (DAC)	4	RGB	15
Drivers	5, 6, 7	Shot noise	3
EEPROM	1, 21	Synchronous mode	22
Flicker noise	16	White noise	16
George Marsaglia	23	Xorshift	23
John von Neumann	22, 29, 31		
Linear congruential generator	23		

