Better Analysis.

USB Explorer 280 Generator

User Manual

Version 1.3
November 19, 2009

Document Revision History

Date	Revision	Changes
July 1, 2009	1.0	Initial release.
Aug. 12, 2009	1.1	Changed Lane to Port in all instructions.
Nov. 5, 2009	1.2	Added new instructions and special registers.
Nov. 19, 2009	1.3	Added description of ref.

Ellisys Contact Details

Ellisys	Phone:	+41 22 777 77 89
Chemin du Grand-Puits 38	Fax:	+41 22 777 77 90
CH-1217 Meyrin Geneva	Email:	info@ellisys.com
Switzerland	Web:	www.ellisys.com

Conditions of Use and Limited Warranty Terms

These conditions and terms are deemed to be accepted by the customer at the time the product is purchased, leased, lent or used, whether or not acknowledged in writing.

Conditions of Use

The customer is only authorized to use the product for its own activities, whether professional or private. Thus, the customer is, in particular, forbidden to resell, lease or lend the product to any third party. In addition, the customer has, in particular, no right to disassembly, modify, copy, reverse engineer, create derivative works from or otherwise reduce or alter the product. The product may also not be used in any improper way.

Limited Warranty Coverage

Ellisys warrants to the original customer of its products that its products are free from defects in material and workmanship for the warranty period. Subject to the conditions and limitations set forth below, Ellisys will, at its option, either repair or replace any part of its products that prove defective by reason of improper workmanship or materials. Repaired parts or replacement products will be provided by Ellisys on an exchange basis, and will be either new or refurbished to be functionally equivalent to new. If Ellisys is unable to repair or replace the product, it will refund the current value of the product at the time the warranty claim is made. In no event shall Ellisys' liability exceed the original purchase price of product.

Excluded Products and Problems

This limited warranty does not cover any damage to this product that results from improper installation, accident, abuse, misuse, natural disaster, insufficient or excessive electrical supply, abnormal mechanical or environmental conditions, or any unauthorized disassembly, repair, or modification. This limited warranty also does not apply to any product on which the original identification information has been altered, obliterated or removed, has not been handled or packaged correctly, or has been sold as second-hand. This limited warranty only applies to the original customer of the product for so long as the original customer owns the product. This limited warranty is non-transferable.

This limited warranty covers only repair, replacement or refund for defective Ellisys products, as provided above. Ellisys is not liable for, and does not cover under warranty, any loss of data or any costs associated with determining the source of system problems or removing, servicing or installing Ellisys products.

Obtaining Warranty Service

To obtain warranty service, you may return a defective product to the authorized Ellisys dealer or distributor from which you purchased the Ellisys product. Please confirm the terms of your dealer's or distributor's return policies prior to returning the product. Typically, you must include product identification information, including model number and serial number with a detailed description of the problem you are experiencing. You must also include proof of the date of original retail purchase as evidence that the product is within the applicable warranty period.

The returned product will become the property of Ellisys. Repaired or replacement product will be shipped at Ellisys' expense. Repaired or replacement product will continue to be covered by this limited warranty for the remainder of the original warranty or 90 days, whichever is longer.

Limitations

THE FOREGOING IS THE COMPLETE WARRANTY FOR ELLISYS PRODUCTS AND SUPERSEDES ALL OTHER WARRANTIES AND REPRESENTATIONS, WHETHER ORAL OR WRITTEN. EXCEPT AS EXPRESSLY SET FORTH ABOVE, NO OTHER WARRANTIES ARE MADE WITH RESPECT TO ELLISYS PRODUCTS AND ELLISYS EXPRESSLY DISCLAIMS ALL WARRANTIES NOT STATED HEREIN, INCLUDING, TO THE EXTENT PERMITTED BY APPLICABLE LAW, ANY WARRANTY THAT MAY EXIST UNDER NATIONAL, STATE, PROVINCIAL OR LOCAL LAW INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTY OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. ALL WARRANTIES, WHETHER EXPRESS OR IMPLIED, ARE LIMITED TO THE PERIODS OF TIME SET FORTH ABOVE. SOME STATES OR OTHER JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES OR LIMITATIONS ON HOW LONG AN IMPLIED WARRANTY LASTS, SO THE ABOVE LIMITATIONS MAY NOT APPLY TO YOU.

ELLISYS PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT EQUIPMENT OR FOR APPLICATIONS IN WHICH THE FAILURE OR MALFUNCTION OF THE PRODUCTS WOULD CREATE A SITUATION IN WHICH PERSONAL INJURY OR DEATH IS LIKELY TO OCCUR. ELLISYS SHALL NOT BE LIABLE FOR THE DEATH OF ANY PERSON OR ANY LOSS, INJURY OR DAMAGE TO PERSONS OR PROPERTY BY USE OF PRODUCTS USED IN APPLICATIONS INCLUDING, BUT NOT LIMITED TO, MILITARY OR MILITARY-RELATED EQUIPMENT, TRAFFIC CONTROL EQUIPMENT, DISASTER PREVENTION SYSTEMS AND MEDICAL OR MEDICAL-RELATED EQUIPMENT.

ELLISYS' TOTAL LIABILITY UNDER THIS OR ANY OTHER WARRANTY, EXPRESS OR IMPLIED, IS LIMITED TO REPAIR, REPLACEMENT OR REFUND. REPAIR, REPLACEMENT OR REFUND ARE THE SOLE AND EXCLUSIVE REMEDIES FOR BREACH OF WARRANTY OR ANY OTHER LEGAL THEORY. TO THE FULLEST EXTENT PERMITTED BY APPLICABLE LAW, ELLISYS SHALL NOT BE LIABLE TO THE CUSTOMER OF AN ELLISYS PRODUCT FOR ANY DAMAGES, EXPENSES, LOST DATA, LOST REVENUES, LOST SAVINGS, LOST PROFITS, OR ANY OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING FROM THE PURCHASE, USE OR INABILITY TO USE THE ELLISYS PRODUCT, EVEN IF ELLISYS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME STATES OR OTHER JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU.

Severability

If any provision or any portion of any provision contained in these terms is held to be invalid, illegal or unenforceable by a court of competent jurisdiction, then the remaining provisions, and if a portion of any provision is unenforceable, then the remaining portion of such provision shall, nevertheless, remain in full force and effect. The parties undertake to negotiate in good faith with a view to replace such invalid, illegal or unenforceable provision or part thereof with another provision not so invalid, illegal or unenforceable with the same or similar effect, and further agree to be bound by the mutually agreed substitute provision.

Warranty Period

The warranty begins on the date of purchase and covers a period of two (2) years.

Governing Law

These conditions and terms shall be governed by and construed in accordance with the law of Switzerland.

Jurisdiction; Venue

The parties consent to the exclusive personal jurisdiction of, and venue in, the District Court of Geneva, Switzerland.

Table of Contents

About this Manual

Typographic Conventions

Bold is used to indicate menu commands, buttons, and tabs.

Italics are used to indicate fields, pane names, window names and cross references.

`Fixed width` is used to indicate system file names, text typed and code snippets.

A warning symbol describes a possible critical situation and how to avoid it.

An information symbol tells you how to respond to a situation that may arise.

A tip symbol tells you information that will help you carry out a procedure.

Where to Find More Help

Go to the Ellisys website and the following pages for the latest information:

- Ellisys products page - Go to **www.ellisys.com/products/** for the latest product information and documentation.
- Application notes and white papers - Go to **www.ellisys.com/technology/** to find up-to-date information about the technology.
- Distributors - Go to **www.ellisys.com/sales/** to find a list of Ellisys distributors.
- Technical support - Go to **www.ellisys.com/support/** to send a question directly to the Ellisys support team.

1. Generator Overview

1.1 Introduction

The Ellisys USB Explorer 280 Generator is an advanced traffic generation and emulation system for SuperSpeed USB 3.0 and USB 2.0 protocols. The Generator verifies product and component functionality, reliability, and performance by generating reproducible traffic patterns, timing scenarios, and various types of errors.

The Generator contains a specialized processor designed specifically for SuperSpeed USB and USB 2.0 protocols, allowing for very fast timing interaction with the connected Device Under Test (DUT). The processor's instruction set enables the user to emulate any SuperSpeed system component, such as a host or device.

The Generator is capable of sending any sequence of packets, link commands, or symbols. The Generator recognizes incoming traffic and can make script-based decisions on this traffic, such as conditional branching or wait states. The Generator's software application enables the user to create, edit, and debug scripts. Traces previously captured by the Ellisys USB Explorer 280 Analyzer can be exported to a script and replayed by the Generator.

1.2 Main Features

The Generator includes the following major features and capabilities:

- Emulate a SuperSpeed Host or Device
- Emulate a USB 2.0 Host or Device
- Perform functional validation and stress testing of protocol stacks
- Inject errors at the physical, link, and protocol layers
- Create scripts from exported analyzer traces

2. Installing the Application

Before installing the software application for the USB Explorer 280 Generator, please ensure the computer system on which it will reside meets the following requirements:

- Microsoft Windows XP SP1 or later.
- Microsoft Windows Installer 3.0 or later. If the installation does not run smoothly, or if the system indicates a version error, update your Windows installer.
- Microsoft .NET Framework version 2.0 or later.
- Intel Core, 1.5 GHz or compatible processor, or better.
- 512 MB RAM or more.
- 1280 x 1024 screen display resolution with 65,536 colors, or better.
- USB 2.0 EHCI Host Controller.

2.1 Software Prerequisites

The USB Explorer 280 Generator requires several software components. Ellisys recommends that you visit the following web pages as needed, to update your versions of Microsoft .NET Framework and Windows:

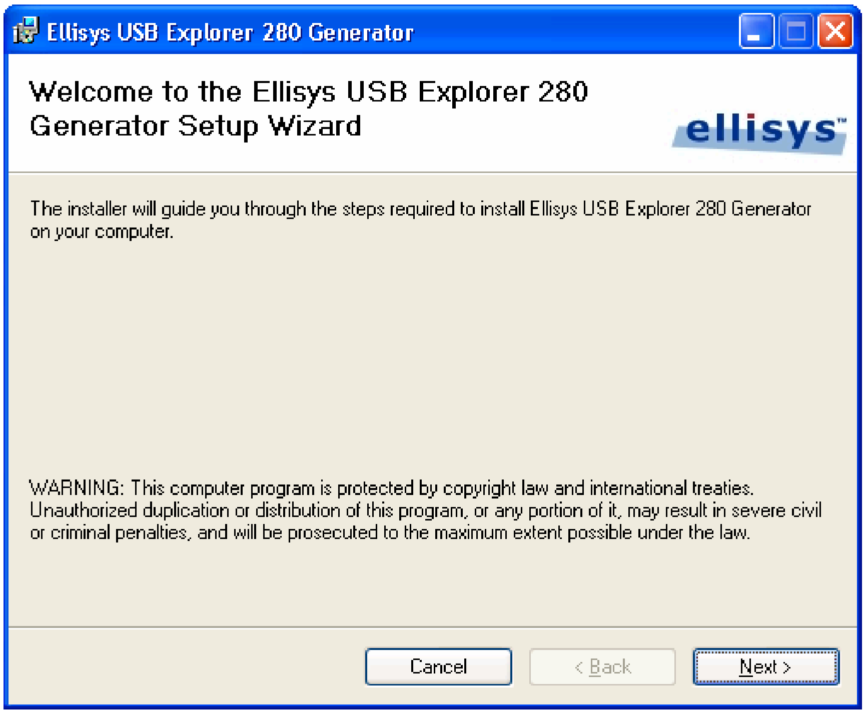
- www.microsoft.com/net to download the Microsoft .NET Framework version 2.0.
- www.update.microsoft.com to update your version of Windows. When using the Windows update service it will automatically download and install the Microsoft .NET Framework version 2.0.

See your system administrator for more information about updating Microsoft .NET Framework and Windows.

2.2 Software Installation

1. Insert the CD-ROM that accompanies the product into the computer's CD-ROM drive.

The Ellisys SuperSpeed USB 280 Generator *Setup Wizard* screen appears:


Ellisys USB Explorer 280 Generator

Welcome to the Ellisys USB Explorer 280 Generator Setup Wizard

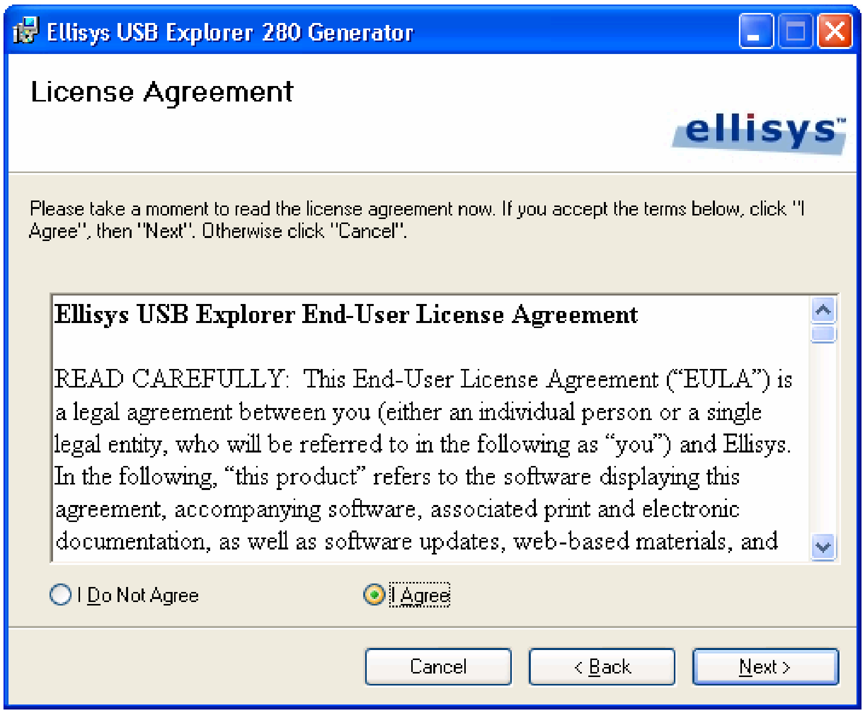
The installer will guide you through the steps required to install Ellisys USB Explorer 280 Generator on your computer.

WARNING: This computer program is protected by copyright law and international treaties. Unauthorized duplication or distribution of this program, or any portion of it, may result in severe civil or criminal penalties, and will be prosecuted to the maximum extent possible under the law.

If the SuperSpeed USB 280 Generator Setup Wizard does not appear, automatically, click the START button on your Windows toolbar, then RUN, and type *d:\setup.exe* (change *d:* to match the drive letter designation of your CD-ROM drive as needed), then click OK.

2. Read the WARNING note and click on **Next**.

The Ellisys USB Explorer 280 *License Agreement* screen appears:



License Agreement

Please take a moment to read the license agreement now. If you accept the terms below, click "I Agree", then "Next". Otherwise click "Cancel".

Ellisys USB Explorer End-User License Agreement

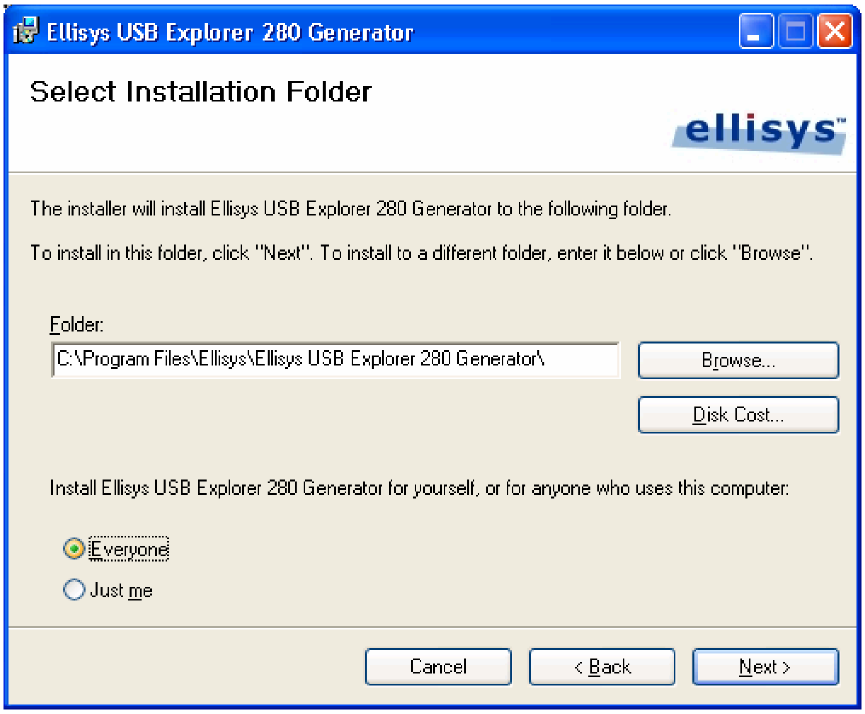
READ CAREFULLY: This End-User License Agreement ("EULA") is a legal agreement between you (either an individual person or a single legal entity, who will be referred to in the following as "you") and Ellisys. In the following, "this product" refers to the software displaying this agreement, accompanying software, associated print and electronic documentation, as well as software updates, web-based materials, and

I Do Not Agree [✓ I Agree]

3. Read the License Agreement carefully, and then select **I Agree**.

4. Click on **Next**.

The *Select Installation Folder* screen appears.



Select Installation Folder

The installer will install Ellisys USB Explorer 280 Generator to the following folder.

To install in this folder, click "Next". To install to a different folder, enter it below or click "Browse".

Folder:
C:\Program Files\Ellisys\Ellisys USB Explorer 280 Generator\

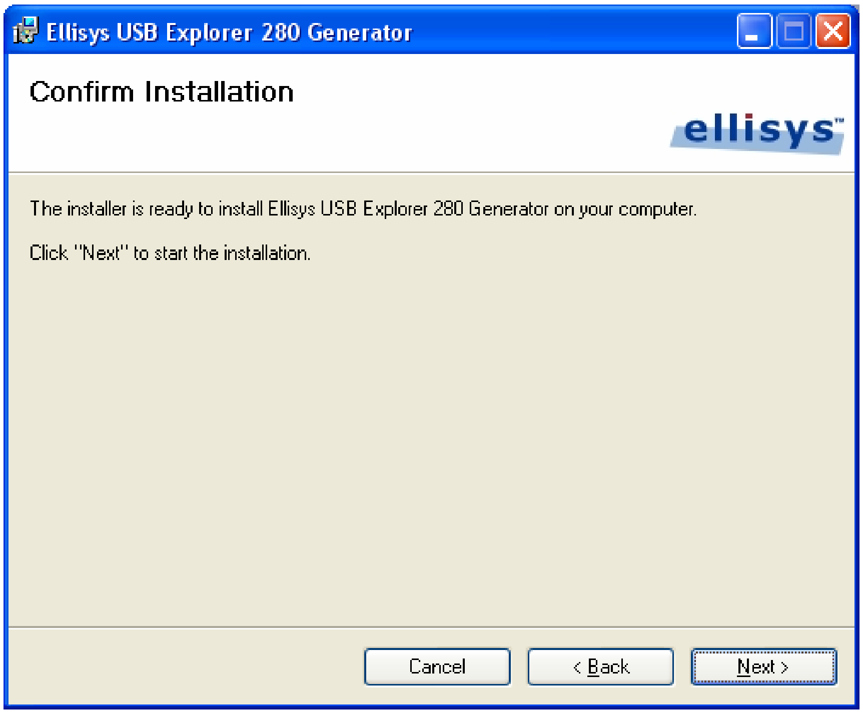
Install Ellisys USB Explorer 280 Generator for yourself, or for anyone who uses this computer:

[✓ Everyone]
Just me

5. The default installation folder appears in the *Folder* field. Ellisys recommends that you use the default folder, however if you wish to change this folder, click on **Browse** and navigate to the folder required.

6. Select whether anyone or only the user currently logged on can access the software by selecting either **Everyone** or **Just me**. Click on **Next**.

The *Confirm Installation* screen appears:

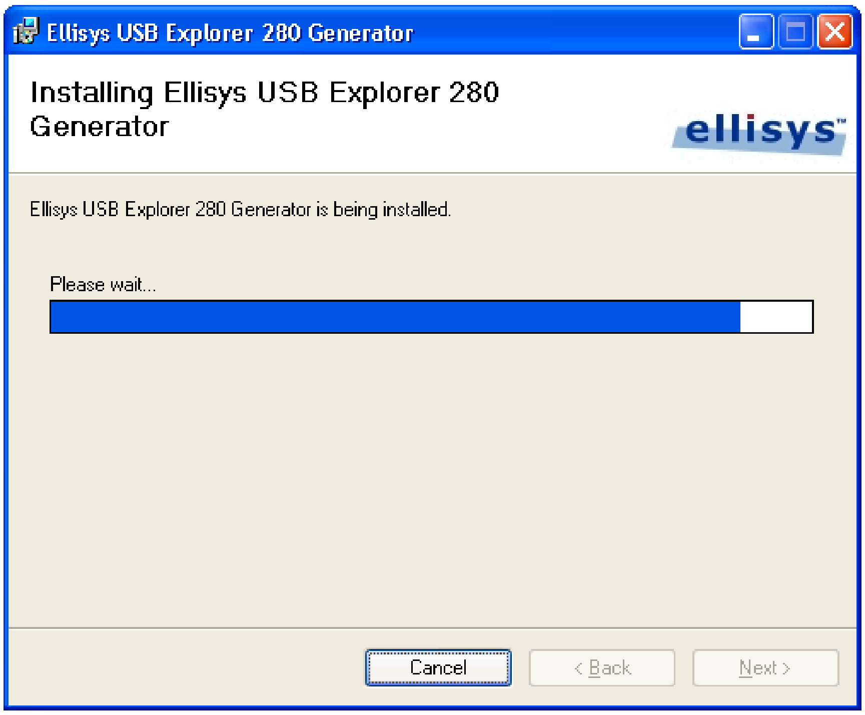


The installer is ready to install Ellisys USB Explorer 280 Generator on your computer.

Click "Next" to start the installation.

7. Click on **Next** to continue the installation.

An Installation Progress screen appears:

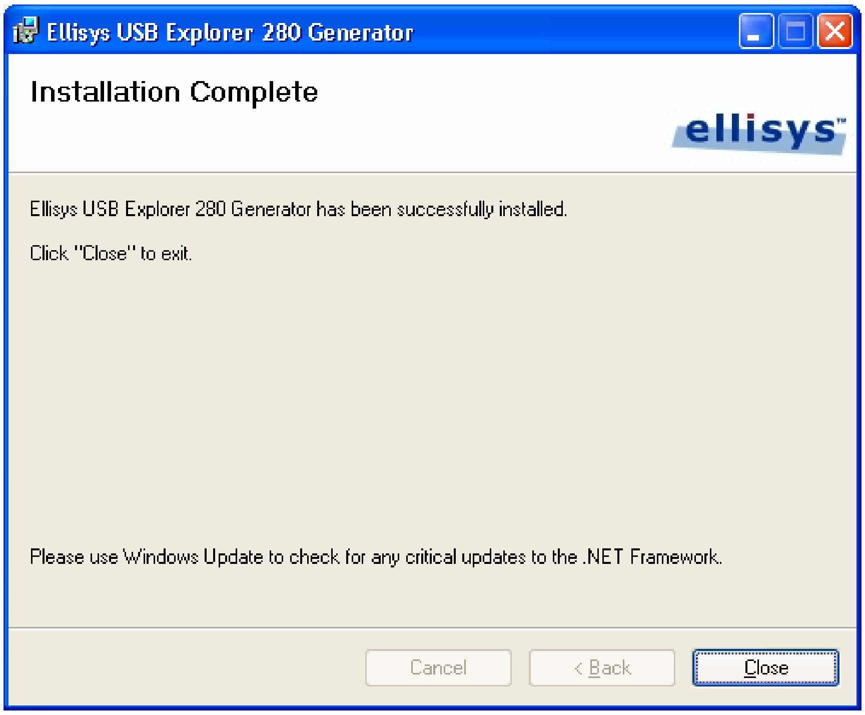


Installing Ellisys USB Explorer 280 Generator

Ellisys USB Explorer 280 Generator is being installed.

Please wait...

When the software has been installed, the *Installation Complete* screen appears:



Installation Complete

Ellisys USB Explorer 280 Generator has been successfully installed.

Click "Close" to exit.

Please use Windows Update to check for any critical updates to the .NET Framework.

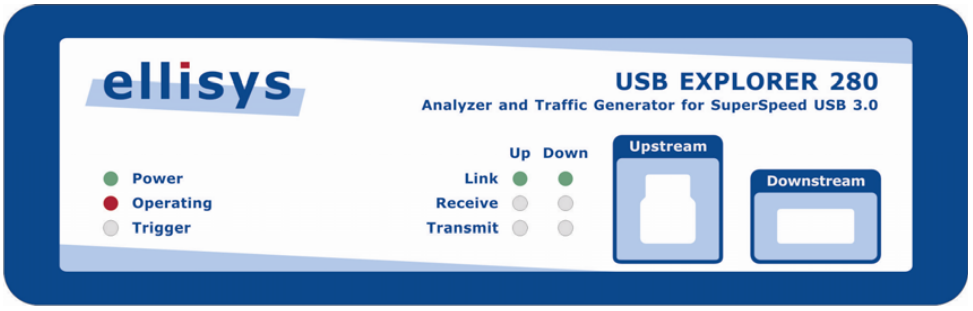
8. Click on **Close**.

The Ellisys USB Explorer 280 Generator is now installed.

After installing the USB Explorer 280 Generator software, a new Hardware Wizard will appear if your units are connected. Refer to section 2.5, *Connecting to the Control Computer*, for more information about installing the USB driver.

2.3 Front Panel Overview

The front panel of the Ellisys USB Explorer 280 Generator is shown below:



When connecting USB cables DO NOT force the connector into the unit. The metal part of the connector should not be inserted completely into the connection port. Forcing the connector or inserting all of the metal part of the connector will break the port connection and is not covered by the warranty.

Upstream Connector

The Upstream Connector is usually used in Device Emulation mode to connect the Generator to a Host Under Test. It can also connect to a Hub Downstream Port for the same purpose.

Downstream Connector

The Downstream Connector is usually used in Host Emulation mode to connect the Generator to a Device Under Test. It can also connect to a Hub Upstream Port for the same purpose.

Power LED

The Power LED indicates if the unit is correctly powered from the supplied 24VDC/2A power adapter and connected to the control computer.

● **Constant green:** Powered and connected, ready to operate.

● **Flashing green:** Powered but not connected.

● **Flashing red:** Connected but not powered.

● **Off:** Not powered and not connected. The Power LED may also be off if when the unit is in power-saving mode after the control computer has been turned off.

Operating LED

The Operating LED indicates if the unit is presently operating or not, for example as protocol analyzer or as traffic generator.

🟢 **Constant green:** Unit is in use.

⚪ **Off:** Unit is not in use and available.

Trigger LED

The Trigger LED indicates input and output trigger events.

🟢 **Flashing green:** Trigger event detected on the input.

🔴 **Flashing red:** Trigger event generated on the output.

⚪ **Off:** No trigger event.

Link LED

The Link LED indicates the status of the generator's upstream and downstream ports. Depending on the generator's state, a port may be used or not. Here are the ports used by the generator depending on the mode:

▪ Host mode:	Only the Downstream port is used.
▪ Device mode:	Only the Upstream port is used.
▪ Hub mode:	Both Upstream and Downstream ports are used.

⚪ **Off:** No receiver detected.

🟠 **Constant orange:** Receiver detected, no SuperSpeed signaling detected.

🟢 **Constant green:** SuperSpeed signaling detected, receiver synchronized.

🔴 **Flashing red:** Link is unstable, frequent loss of synchronization.

Receive LED

The Receive LED indicates if payload (Data Packets) or errors (CRC, invalid symbols) are received on a given port.

Off: No payload or errors detected.

Flashing green: Payload detected.

Flashing red: Errors detected.

Transmit LED

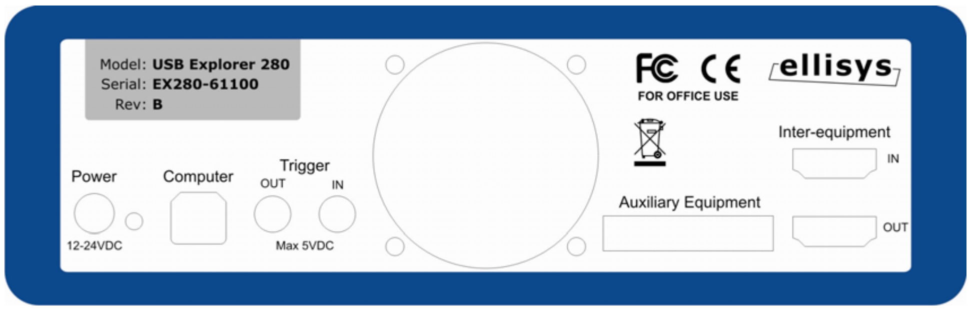
The Transmit LED indicates if payload (DPP) or errors (CRC, invalid symbols) are transmitted on a given port.

Off: No data sent.

Flashing green: Data Packet sent.

2.4 Back Panel Overview

The back panel of the Ellisys USB Explorer 280 Generator is shown below:



When connecting the USB cable DO NOT force the connector into the unit. The metal part of the connector should not be inserted completely into the connection port. Forcing the connector or inserting all of the metal part of the connector will break the port connection and is not covered by the warranty.

Power

DC jack power input. The nearby LED illuminates constant green if a correct voltage is applied, and illuminates constant red if the voltage is applied reversed.

Accepted Voltage Range: 12V to 24V DC.
Minimum Power: 18W

Computer

Type B USB 2.0 connector. Attaches to control computer.

Trigger OUT

SMA connector used for sending TTL voltage level shift or pulse to external equipment.

Trigger IN

SMA connector used for accepting TTL voltage level shift or pulse from external equipment.

Auxiliary Equipment

Reserved for future extensions.

Inter-equipment

Reserved for future extensions.

2.5 Connecting to the Control Computer

The USB Explorer 280 Generator is controlled over a high-speed USB 2.0 connection by a PC hosting the Generator application, enabling the use of any notebook or desktop computer. The USB driver must be installed before the Generator can be used.

Although the unit can upload or download data on a full speed USB 1.1 connection, Ellisys strongly recommends that you connect it to a high speed USB 2.0 port to obtain optimal performance. If you experience problems with the USB Explorer 280, please ensure it is connected on a high speed USB 2.0 enabled host controller before contacting technical support.

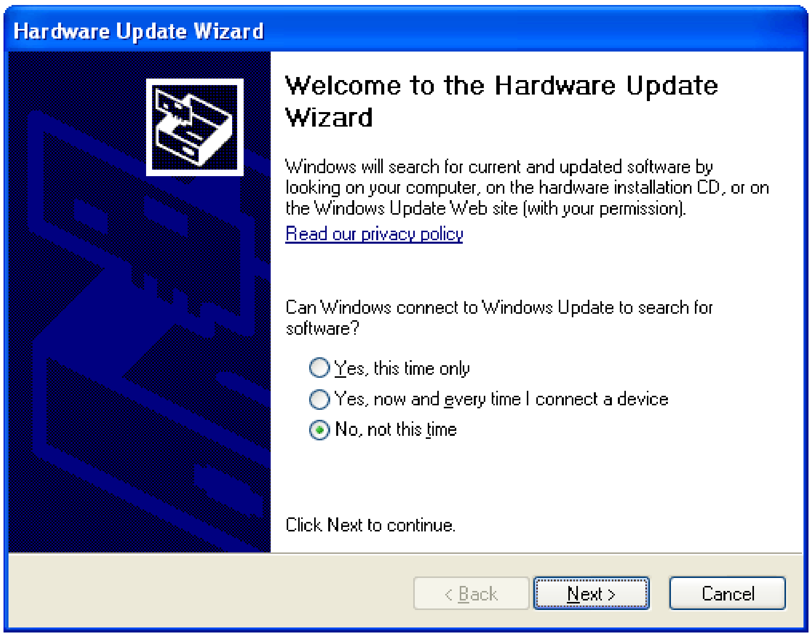
Follow the steps below to install the USB driver:

1. Connect a USB 2.0 cable between the Type B USB receptacle Generator back panel and the PC. If attaching the Generator for the first time, wait until Windows displays a message indicating that a new device has been found (typically a small bubble indication at the lower-right of the screen), then go to step 3.

2. If you want to update a previously installed device driver:

   - Open the Device Manager: Start | Control Panel
   - Double-Click the System icon.
   - Click on the Hardware tab.
   - Click on Device Manager.
   - Click on Ellisys protocol analyzers.
   - Right-click and select Update Driver.

The *Hardware Update Wizard* opens:


Hardware Update Wizard

Welcome to the Hardware Update Wizard

Windows will search for current and updated software by looking on your computer, on the hardware installation CD, or on the Windows Update Web site (with your permission).
Read our privacy policy

Can Windows connect to Windows Update to search for software?

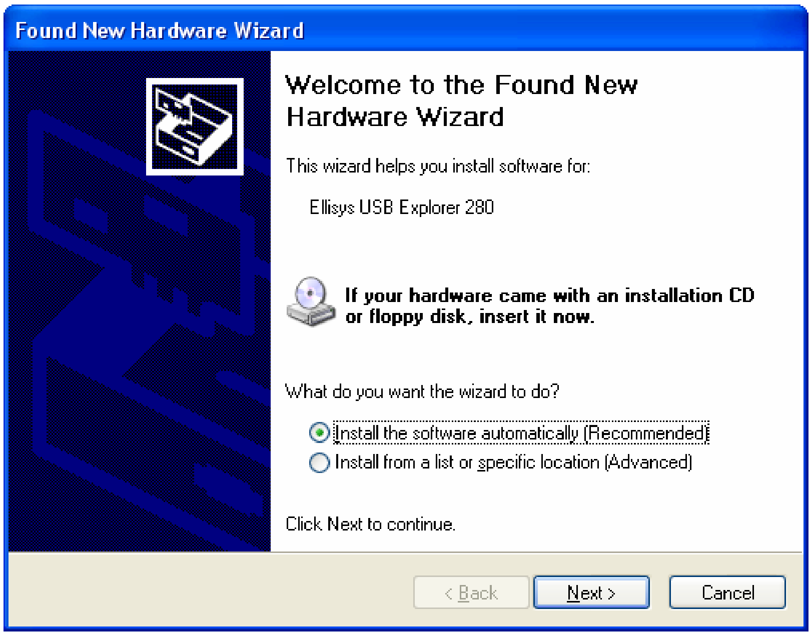
- ○ Yes, this time only
- ○ Yes, now and every time I connect a device
- [✓ No, not this time]

Click Next to continue.

3. Select **No, not this time**.

4. Click on **Next**.

The *Found New Hardware Wizard* appears:


Found New Hardware Wizard

Welcome to the Found New Hardware Wizard

This wizard helps you install software for:

Ellisys USB Explorer 280

If your hardware came with an installation CD or floppy disk, insert it now.

What do you want the wizard to do?

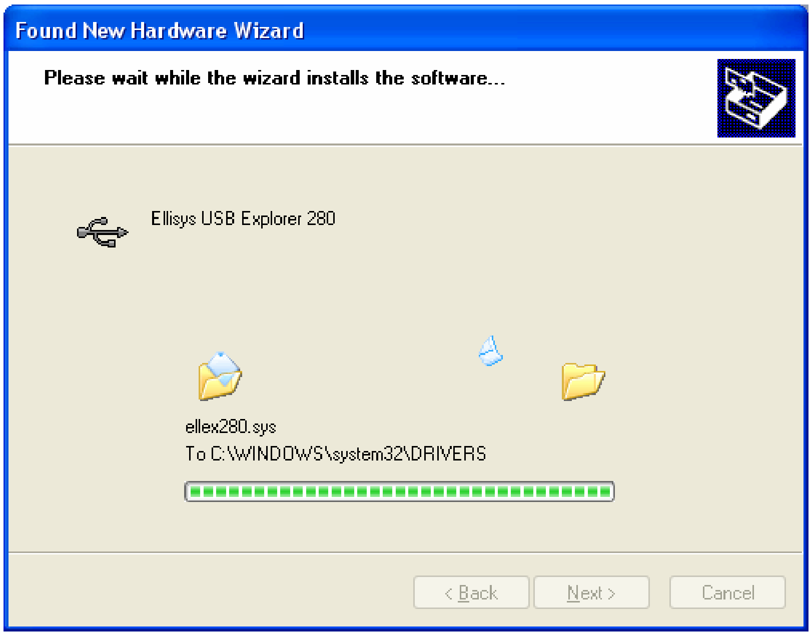
- [✓ Install the software automatically (Recommended)]
- ○ Install from a list or specific location (Advanced)

Click Next to continue.

5. Select **Install the software automatically (Recommended).**

6. Click on **Next**.

The *Please wait while the wizard installs the software* window appears:



Found New Hardware Wizard

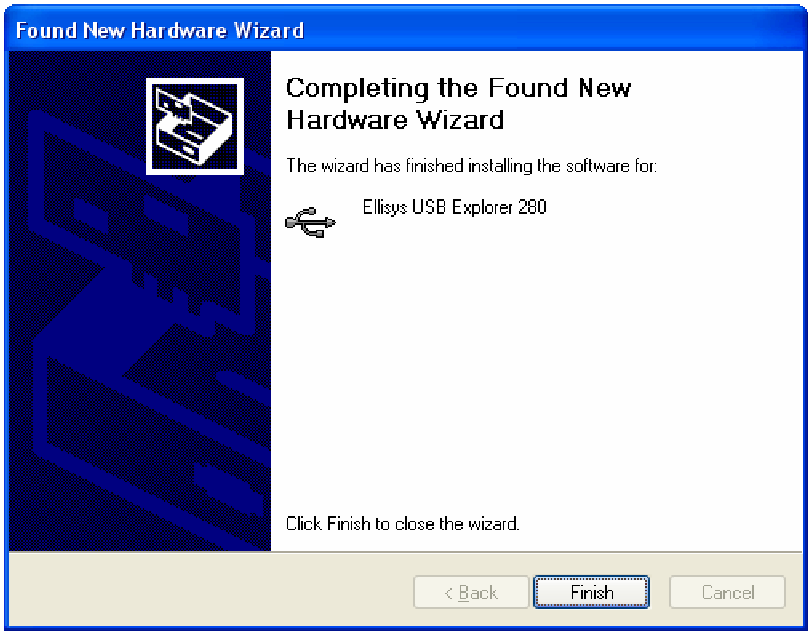
Please wait while the wizard installs the software...

Ellisys USB Explorer 280

ellex280.sys
To C:\WINDOWS\system32\DRIVERS

Windows now installs the driver.

7. When the installation is complete, the *wizard has finished installing the software* screen appears:



Found New Hardware Wizard

Completing the Found New Hardware Wizard

The wizard has finished installing the software for:

Ellisys USB Explorer 280

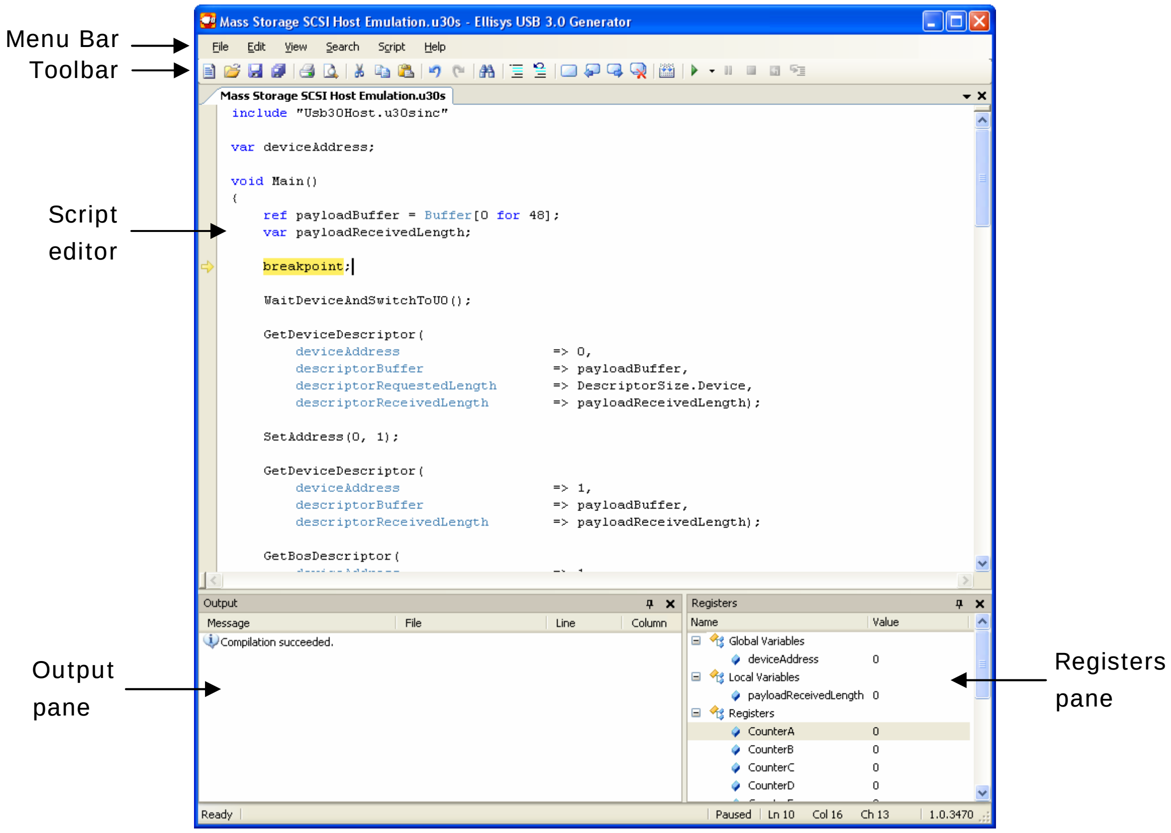
Click Finish to close the wizard.

8. Click on **Finish**.

The installation is complete.

3. User Interface Reference

The user interface of the USB Explorer 280 Generator application provides various panes, menus, toolbars, and other visual elements.


Menu Bar
Toolbar
Script editor
Output pane
Registers pane
Mass Storage SCSI Host Emulation.u30s - Ellisys USB 3.0 Generator
File
Edit
View
Search
Script
Help
Mass Storage SCSI Host Emulation.u30s

```
include "Usb30Host.u30sinc"

var deviceAddress;

void Main()
{
    ref payloadBuffer = Buffer[0 for 48];
    var payloadReceivedLength;

    breakpoint;

    WaitDeviceAndSwitchToU0();

    GetDeviceDescriptor(
        deviceAddress                   => 0,
        descriptorBuffer                => payloadBuffer,
        descriptorRequestedLength       => DescriptorSize.Device,
        descriptorReceivedLength        => payloadReceivedLength);

    SetAddress(0, 1);

    GetDeviceDescriptor(
        deviceAddress                   => 1,
        descriptorBuffer                => payloadBuffer,
        descriptorReceivedLength        => payloadReceivedLength);

    GetBosDescriptor(
```

Output

Message	File	Line	Column
Compilation succeeded.			

Registers

Name	Value
Global Variables	
deviceAddress	0
Local Variables	
payloadReceivedLength	0
Registers	
CounterA	0
CounterB	0
CounterC	0
CounterD	0

Ready | Paused | Ln 10 | Col 16 | Ch 13 | 1.0.3470

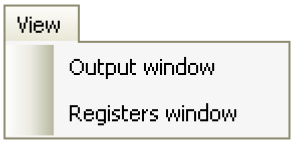
The Generator application has several default panes. Each pane displays specific information or allows the user to interact with the software for a given task:

- ***Script Editor*** – Shows the current script. The *Script Editor* allows for editing the script, setting or clearing breakpoints, and placement of bookmarks to enhance navigation through the script.
- ***Output Pane*** – Shows messages about the script after compiling. If there is an error in the script, the *Output* pane will show an error description and the error's location (file, line, and column).
- ***Register Pane*** – Shows the contents of variables used in the script. Refer to section 3.14, *Working with Registers*, for more information.

3.1 Organizing Panes

To open or display a pane:

1. Select **View** in the menu and select the desired pane.



The selected pane opens.

To close a pane:

1. Click on **Close** ✕ positioned at the top-right corner of the title bar of the pane.

The pane closes.

To hide a pane:

1. Click on **Auto-Hide** positioned at the top-right corner of the title bar.

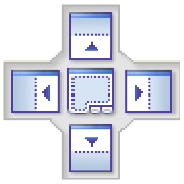
The pane is hidden and the pane's name now appears as a tab at the side of the screen.

To move a pane or a window:

1. Click on the title bar of the desired pane or window.

2. Depress and hold the left mouse button and drag the pane or window.

A window placer appears:



3. Keep the mouse button depressed and point to one of the following:

- **Center** to open a pane as a floating window in the screen.
- **Top** to move the pane to the top of the screen or pane group.
- **Right** to move the pane to the right of the screen or pane group.
- **Left** to move the pane to the left of the screen or pane group.
- **Bottom** to move the pane to the bottom of the screen or pane group.

3.2 Main Toolbar

The table below shows the USB Explorer 280 Generator toolbar buttons and their actions:

New document	Opens a new document.
Open document	Opens a folder to open a previously saved document.
Save document	Saves a document.
Print	Opens Print Options for printing a document.
Print Preview	Opens the Print Preview window.
Cut	Cuts a text selection.
Copy	Copies a text selection.
Paste	Pastes a selection of copied or cut text.
Undo	Undoes the previous action.
Redo	Redoes the previous action.
Find/Replace	Opens the find and replace window.
Comment Selection	Comments out one or more lines.
Uncomment Selection	Uncomments one or more lines.
Toggle Bookmark	Toggles a bookmark at a selected line.
Previous Bookmark	Finds the previous bookmark.
Next Bookmark	Finds the next bookmark.
Clear Bookmarks	Clears all bookmarks.
Compile	Compiles a script.
Run	Runs a stopped or paused script.
Break	Pauses a running script.
Stop	Stops a running script.
Restart	Stops and restarts a script from the beginning.
Step	Steps from line to line in the script.

3.3 Main Menu

The table below shows the SuperSpeed USB 280 Generator main menu options and their actions, with shortcuts shown in parentheses:

File

New (CTRL+N)	Opens a new document.
Open (CTRL+O)	Opens a folder to open a previously saved document.
Save (CTRL+S)	Saves a document.

Save All	Saves all documents currently open.
Save As (CTRL+SHFT+S)	Saves a file with a new name.
Load Sample	Opens sample files provided with application.
Load Library	Opens include libraries.
Page Setup	Opens Page Setup dialog allowing user to set page margins and other parameters.
Print Preview	Opens the Print Preview window.
Print (CTRL+P)	Opens Print Options for printing a document.
Exit	Closes the application.

Edit

Undo (Ctrl+Z)	Undoes the previous action.
Redo (Ctrl+Y)	Redoes the previous action.
Cut (Ctrl+X)	Cuts a text selection.
Copy (Ctrl+C)	Copies a text selection.
Paste (Ctrl+V)	Pastes a selection of copied or cut text.

Edit | Advanced

Mark Line Modifications	Marks line modifications in the script.
Highlight Current Line	Highlights the current line in the script.
Show Column 80 Guide	Displays the column guide in the script.
Comment Selection (CTRL+K, CTRL+C)	Adds a comment to the currently selected line.
Uncomment Selection (CTRL+K, CTRL+U)	Removes the comment from the selected line.
Make Uppercase (CTRL+SHFT+U)	Changes selected lowercase text to uppercase.
Make Lowercase (CTRL+SHFT+U)	Changes selected uppercase text to lowercase.

Edit | Bookmarks

Toggle Bookmark (CTRL+K, CTRL+K)	Toggles a bookmark at a selected line.
Enable Bookmark (CTRL+K, CTRL+N)	Enables the selected bookmark.
Previous Bookmark (CTRL+K, CTRL+P)	Finds the previous bookmark.
Next Bookmark (CTRL+K, CTRL+L)	Finds the next bookmark.
Clear Bookmarks (CTRL+K, CTRL+H)	Clears all bookmarks.
Insert Code Snippet (CTRL+I)	Opens a dialog permitting insertions of pre-defined code snippets into the script at the currently selected line.

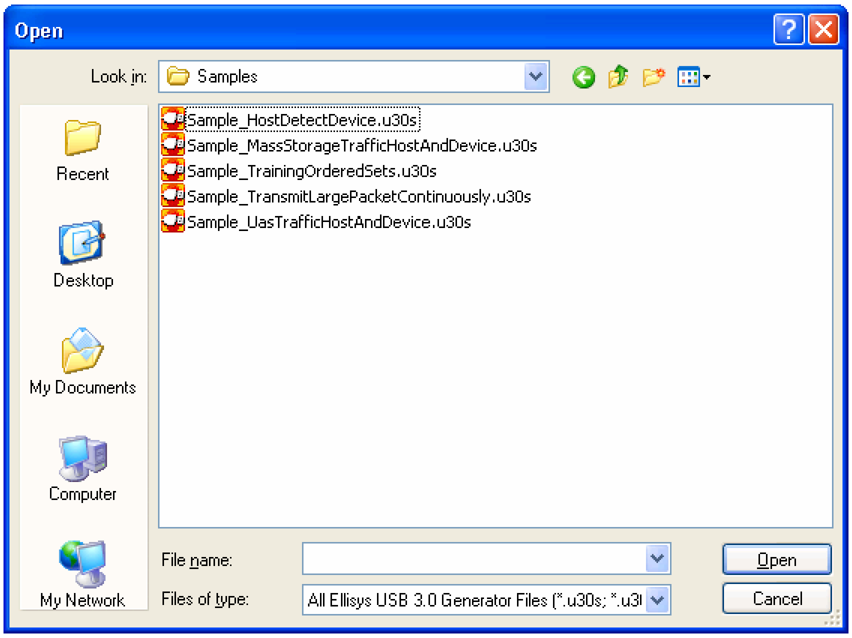
View	
Output Window	Opens or closes the Output window.
Registers Window	Opens or closes the Registers window.
Search	
Find (CTRL+F)	Opens the Find window.
Replace (CTRL+H)	Opens the Replace window.
Find Next (F3)	Searches forward to find the text previously entered into the Find window.
Find Previous (SHFT+F3)	Searches backward to find the text previously entered into the Find window.
Go To Line (CTRL+G)	Opens the Go To Line window.
Script	
Compile (F7)	Compiles a script.
Run (F5)	Runs a stopped or paused script.
Break	Pauses a script when running.
Stop (SHFT+F5)	Stops a running script.
Restart	Stops and restarts a script from the beginning.
Step (F10)	Steps from line to line in a script.
Toggle Breakpoint (F9)	Toggles a breakpoint at a selected line.
Clear All Breakpoints (CTRL+SHFT+F9)	Removes all breakpoints in a script.
Select a Generator	Opens the Available Generators window.
Help	
Ellisys website	Opens the Ellisys website in the default browser.
Contact support	Opens a form in the default browser to contact Ellisys technical support.
Check for updates	Checks online for the latest software version.
About	Opens the About window.

3.4 Opening a File

To open a file:

Select **File |Open** in the menu or click on **Open Document.**

The Open File menu appears:



Sample_HostDetectDevice.u30s
Sample_MassStorageTrafficHostAndDevice.u30s
Sample_TrainingOrderedSets.u30s
Sample_TransmitLargePacketContinuously.u30s
Sample_UasTrafficHostAndDevice.u30s

All Ellisys USB 3.0 Generator Files (*.u30s; *.u3l

1. Select the file required and click **Open**.

The selected file opens in the software.

3.5 Saving a File

To save a file:

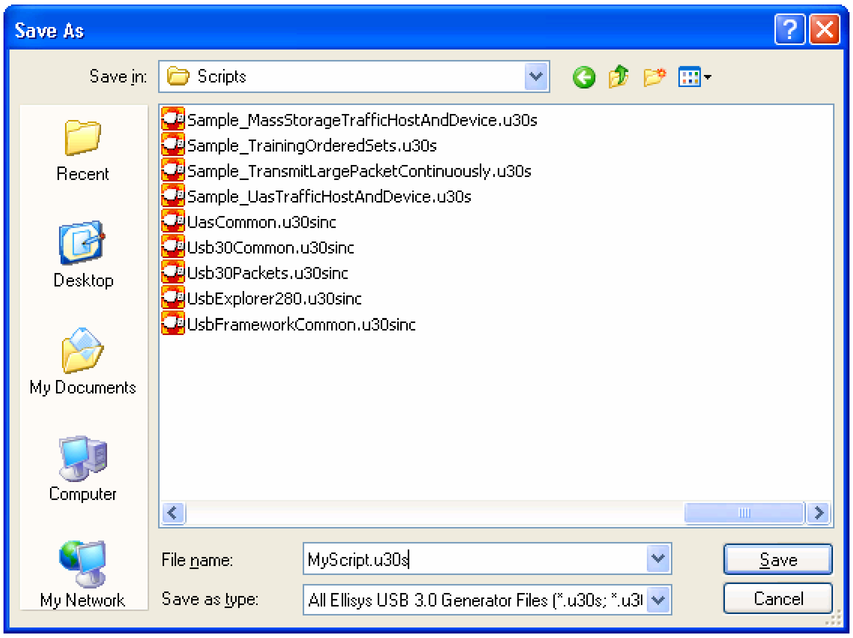
1. Select **File |Save** in the menu or click on **Save Document.**

The file is saved.

To save a file with a new name:

Select **File |Save As** in the menu**.**

The Save As menu appears:



Sample_MassStorageTrafficHostAndDevice.u30s
Sample_TrainingOrderedSets.u30s
Sample_TransmitLargePacketContinuously.u30s
Sample_UasTrafficHostAndDevice.u30s
UasCommon.u30sinc
Usb30Common.u30sinc
Usb30Packets.u30sinc
UsbExplorer280.u30sinc
UsbFrameworkCommon.u30sinc

File name: MyScript.u30s
Save as type: All Ellisys USB 3.0 Generator Files (*.u30s; *.u3

1. Navigate to the directory where the file is to be saved.

2. Enter the desired name of the file in the *File name* field and click on **Save**.

The file is saved with the modified name and the original file is not modified.

3.6 Printing a File

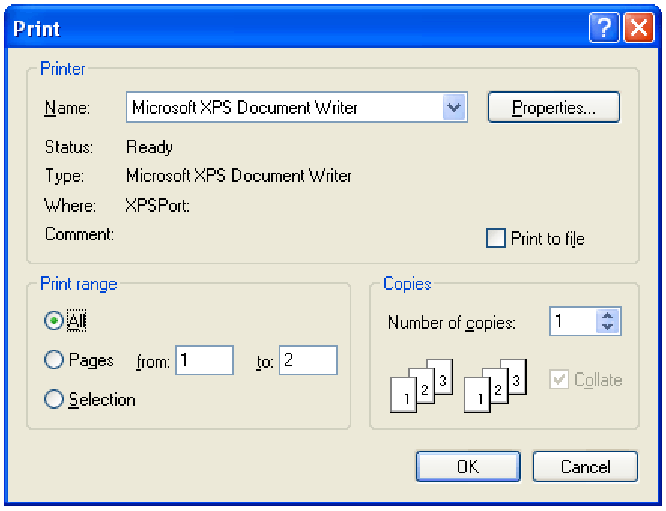
Use the Page Setup option, **File | Page Setup**, to setup how the file should be printed. This option will depend on the printer, please see your printer's documentation for more information.

A file can be very large therefore it is advisable to check the size of the file before trying to print the file.

To print a file:

1. Select **File |Print** in the menu or click on **Print.**

The *Print* window appears:



2. Select the printer and printer setup if required.

3. Click on **OK**.

The file is printed.

3.7 Editing a Script

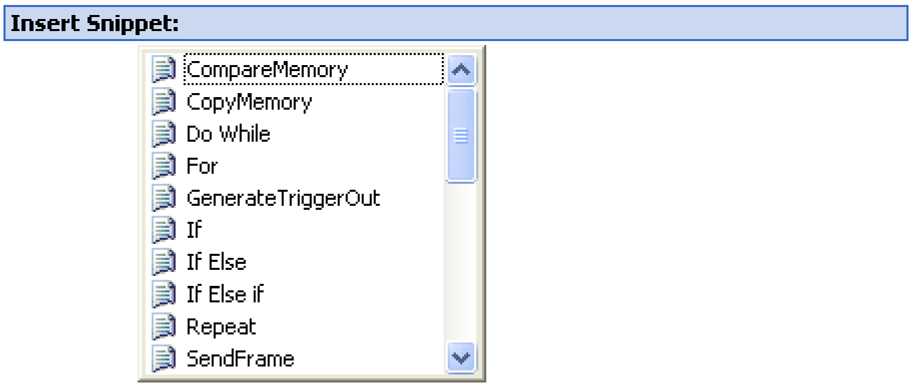
The SuperSpeed Explorer 280 Generator includes several specialized instructions. Example code for these instructions can be inserted to assist in writing scripts. An example code is called a code snippet. A full description of specialized instructions can be found in Chapter 5, *Hardware Instructions Set Reference*.

To insert a code snippet:

1. Click on the point in the script where the code snippet is to be inserted.

2. Select **Edit | Insert Code Snippet** in the menu (or press CTRL+I).

The *Code Snippet* list appears:



3. Select the desired code snippet from the list.

4. Double-click on the desired code snippet or select the desired code snippet and press ENTER.

The selected code snippet is inserted into the script and can be modified as needed.

3.8 Advanced Editing Features

All advanced editing features for the USB Explorer 280 can be accessed by clicking **Edit | Advanced** in the menu.

To mark or unmark line modifications:

1. Select **Edit | Advanced | Mark Line Modifications** in the menu.

All lines that have been modified are marked with a yellow highlight.

To highlight the current line:

1. Select **Edit | Advanced | Highlight Current Line** in the menu.

The line with the cursor is highlighted.

To display the column 80 guide:

1. Select **Edit | Advanced | Show Column 80 Guide** in the menu.

To comment a selection in the script:

1. Select the lines that will be commented.

2. Click on Comment Selection or select **Edit | Advanced | Comment Selection** in the menu.

Comment markers are inserted before the selected lines.

To uncomment a selection in the script:

1. Select the commented lines desired to be uncommented.

2. Click on Uncomment Selection or select **Edit | Advanced | Uncomment Selection** in the menu.

Comment markers are removed from the selected lines.

To change text case:

1. Select the desired text in the script.

2. To change lowercase to uppercase, select **Edit | Advanced | Make Uppercase** to change the text's case from lowercase to uppercase or press CTRL+SHFT+U.

or

3. To change uppercase to lowercase, select **Edit | Advanced | Make Lowercase** to change the text's case from lowercase to uppercase or press CTRL+U.

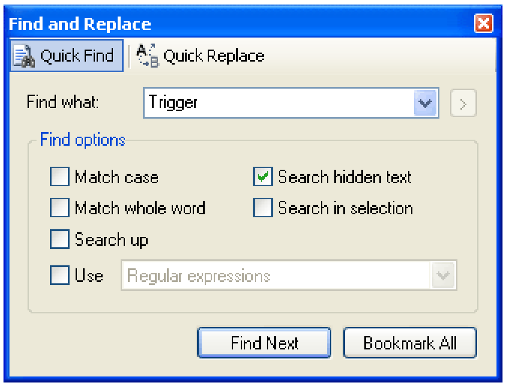
3.9 Searching

Search, find, and replace options can be accessed by clicking Search in the menu.

To search text:

Click on Find/Replace or select Search **| Find** in the menu or press **CTRL+F.**

The Find/Replace menu appears:



1. Enter the desired information in the *Find what* field.

or

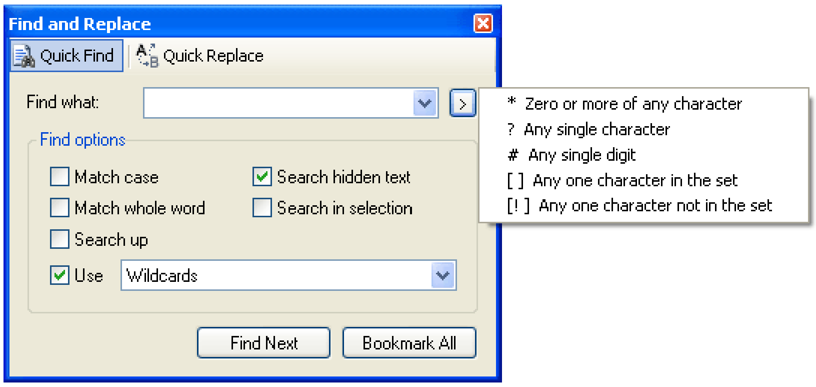
2. Select the **Use** box to use Regular expressions or Wildcards.

Regular expressions or Wildcards can be selected as an option.

3. If the **Use** box is checked, select *Regular expressions* or *Wildcards* from the drop-down list. The **Right Arrow** beside the *Find what* field becomes enabled.

4. Click on the **Right Arrow** .

If *Wildcards* has been selected from the **Use** drop-down list, a *Wildcard list* appears:

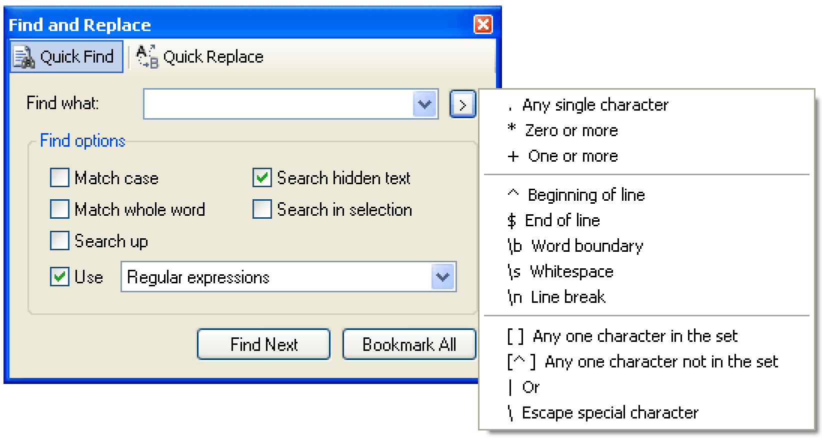


Match case [ ] · Search hidden text [✓] · Match whole word [ ] · Search in selection [ ] · Search up [ ] · Use [✓] Wildcards

*	Zero or more of any character
?	Any single character
#	Any single digit
[ ]	Any one character in the set
[!]	Any one character not in the set

5. Select the Wildcard desired.

If *Regular expressions* has been selected from the **Use** drop-down list, a *Regular expressions list* appears:



Match case [ ] · Search hidden text [✓] · Match whole word [ ] · Search in selection [ ] · Search up [ ] · Use [✓] Regular expressions

.	Any single character
*	Zero or more
+	One or more
^	Beginning of line
$	End of line
\b	Word boundary
\s	Whitespace
\n	Line break
[ ]	Any one character in the set
[^]	Any one character not in the set
|	Or
\	Escape special character

6. Select the Regular expression desired.

7. Select the desired *Find Options* check boxes.

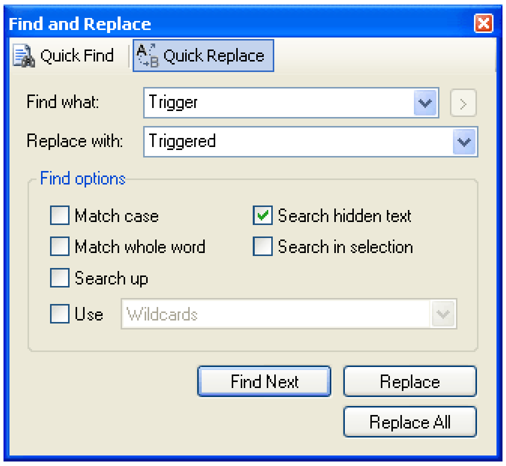
8. Click on **Find Next** to find the next occurrence or click on **Bookmark All** to bookmark all occurrences.

The selected search is performed.

To replace text:

1. Click on **Find / Replace** and then click on **Quick Replace**
   or
   Select **Search | Replace** in the menu
   or
   press CTRL+H.

The *Find/Replace* menu appears:



Find what: Trigger
Replace with: Triggered
Find options
☐ Match case
☐ Match whole word
☐ Search up
☐ Use Wildcards
✓ Search hidden text
☐ Search in selection

2. Enter the desired search text the *Find what* field.

3. Enter the replacement text in the *Replace with* field.

4. Select the desired *Find options* check boxes.

5. Click on **Find Next** to find the next occurrence or click on **Replace** or **Replace All** to replace the next occurrence or all occurrences.

The selected replacement is performed.

3.10 Working with Bookmarks

A bookmark is a useful tool that allows for marking lines of code to assist the user in navigating through the script.

All bookmark options can be accessed by selecting **Edit | Bookmarks** in the menu.

To toggle a bookmark:

1. Select a line where the bookmark is to be inserted.

Click on Toggle Bookmark
or
Select **Edit | Bookmarks | Toggle Bookmark** in the menu**.**

The bookmark is inserted beside the selected line.

To enable a bookmark:

1. Click on the line beside the bookmark.

Select **Edit | Bookmarks | Enable Bookmark** in the menu**.**

The selected bookmark is enabled.

To move to the next or previous bookmark:

1. Click on **Next Bookmark**
   or
   Select **Edit | Bookmarks | Next Bookmark** in the menu.

A flashing cursor appears beside the next bookmark.

Click on **Previous Bookmark**
or
Select **Edit | Bookmarks | Previous Bookmark** in the menu**.**

A flashing cursor appears beside the previous bookmark.

To remove all bookmarks:

1. Click on **Clear Bookmarks**
   or
   Select **Edit | Bookmarks | Clear Bookmarks** in the menu**.**

All bookmarks in the script are removed.

3.11 Working with Breakpoints

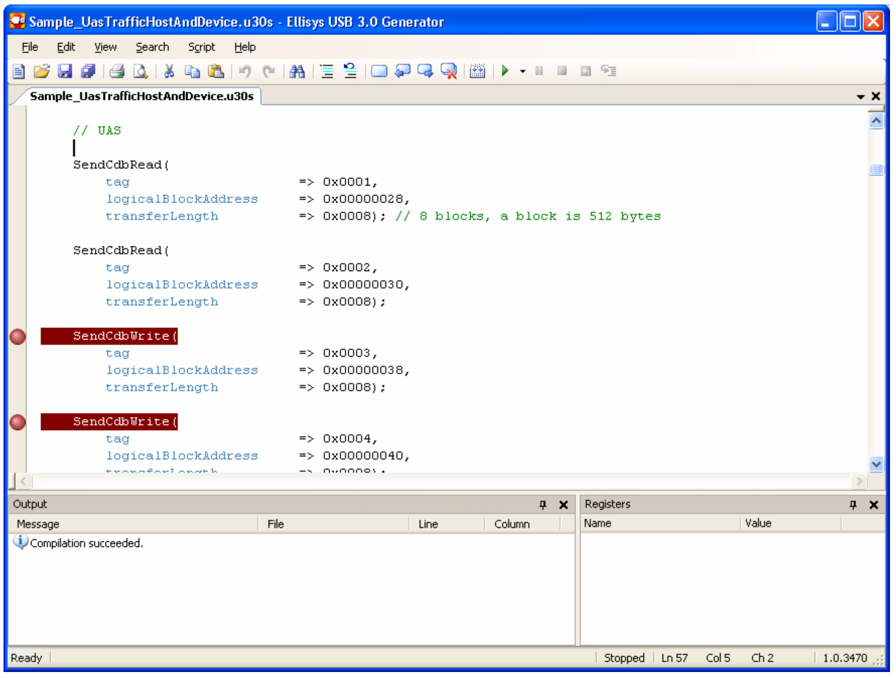
A breakpoint is a point in a program which is used to temporarily halt the execution of that program.

To insert a breakpoint:

1. Select a line where the breakpoint is to be inserted.

2. Select **Script | Toggle Breakpoint** in the menu
   or
   Press F9.

A breakpoint is inserted beside the selected line.


Sample_UasTrafficHostAndDevice.u30s - Ellisys USB 3.0 Generator
File
Edit
View
Search
Script
Help
Sample_UasTrafficHostAndDevice.u30s

```
// UAS

SendCdbRead(
    tag                    => 0x0001,
    logicalBlockAddress    => 0x00000028,
    transferLength         => 0x0008); // 8 blocks, a block is 512 bytes

SendCdbRead(
    tag                    => 0x0002,
    logicalBlockAddress    => 0x00000030,
    transferLength         => 0x0008);

SendCdbWrite(
    tag                    => 0x0003,
    logicalBlockAddress    => 0x00000038,
    transferLength         => 0x0008);

SendCdbWrite(
    tag                    => 0x0004,
    logicalBlockAddress    => 0x00000040,
```

Output

Message	File	Line	Column
Compilation succeeded.			

Registers

Name	Value

Ready | Stopped | Ln 57 | Col 5 | Ch 2 | 1.0.3470

To remove all breakpoints:

1. Select **Script | Clear All Breakpoints** in the menu
   or
   Press CTRL+SHFT+F9.

All breakpoints in the script are removed.

3.12 Compiling a Script

To compile a script:

1. Open a script file as described in section 3.4, *Opening a File*
   or
   Open a new script file and save it.

2. Click on Compile
   or
   Select **Script| Compile** in the menu.

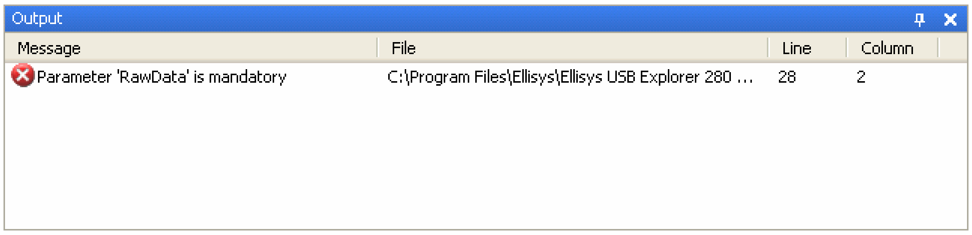
The USB Explorer 280 compiles the script.

If the compilation is successful, a "*Compilation Succeeded*" message will appear in the *Output* pane. If the compilation is unsuccessful, error messages will appear in the *Output* pane.

To find an error in a compiled script:

1. Compile a script as described above.

Compilation errors are listed in the *Output* pane under the *Message* column:



Output

Message	File	Line	Column
Parameter 'RawData' is mandatory	C:\Program Files\Ellisys\Ellisys USB Explorer 280 ...	28	2

2. Double-click on the desired error message in the *Output* pane.

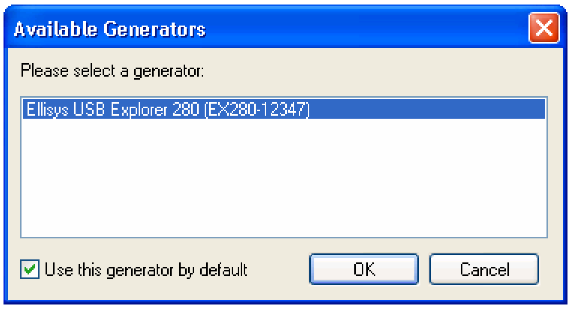
The line that contains the errors is highlighted in the main script pane.

3.13 Running a Script

To select a generator:

1. Select **Script | Select a Generator** in the menu.

The *Available Generators* menu appears:


Available Generators
Please select a generator:
Ellisys USB Explorer 280 (EX280-12347)
Use this generator by default
OK
Cancel

2. Select the desired generator, and click on **OK**.

It is advisable to select a generator as the default generator by click the **Use this generator by default** check box. This will stop the *Available Generators* dialog from appearing every time the software is run.

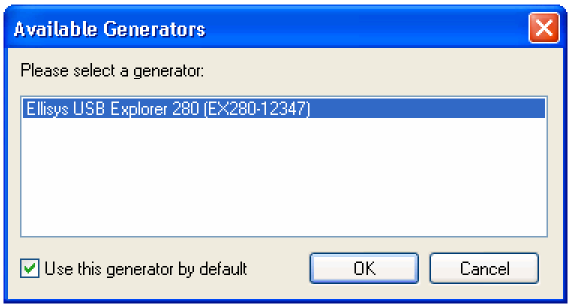
The generator is selected.

To run a script:

1. Open a script file
   or
   Create a new script file and save it

2. Click on **Run**
   or
   Select **Script | Run** in the menu.

If a generator was not selected as a default generator, then the *Available Generators* menu appears:


Available Generators
Please select a generator:
Ellisys USB Explorer 280 (EX280-12347)
Use this generator by default
OK
Cancel

3. Select the desired generator and click **OK**.

The script runs using the selected generator.

To break or pause a script:

1. Run a script as described in section 3.13, *Running a Script*.

2. Click on **Break**
   or
   Select **Script | Break** in the menu.

The script is paused.

To stop a script:

1. Run a script.

2. Click on Stop
   or
   Select **Script | Stop** in the menu.

The script is stopped.

To restart a script:

1. Click on **Restart**
   or
   Select **Script | Restart** in the menu.

The script is restarted.

To step a script:

1. Click on **Step**
   or
   Select **Script | Step** in the menu
   or
   Press F10

The script is run command by command.

3.14 Working with Registers

This section describes the usage of registers within a script. For more information about registers, refers to section 0,

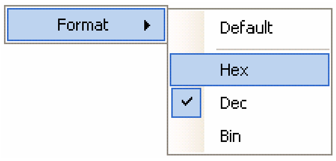
Counters.

All registers are displayed in the *Registers* pane.

To select a register format:

1. Right-click on one of the *registers* in the Registers pane.

The *Format* submenu appears:



2. Click on the desired numbering format, **Dec**, **Hex**, or **Bin**.

The register numbering format is changed as selected and any values present are updated in the selected format.

4. Language Reference

4.1 Comments

Single line comments are done using the // characters.

```
void Main()
{
   // This is a single line comment
   CopyMemory(Src => [ 0x00, 0x00 ], Dst => Buffer, DstOffset => 200);
}
```

Multi-Line comments are opened using /* characters, and closed using the */ characters.

```
void Main()
{
   /* This is a multi line comment, which prevents
      the following instruction from being executed:
   CopyMemory(Src => [ 0x00, 0x00 ], Dst => Buffer, DstOffset => 200);
   */
}
```

4.2 Include Files

Files can be included using the include directive.

The example below shows a script that calls an include file, which uses the macro declared inside the file:

```
include "MyInclude.u30sinc"

void main()
{
   // Calls a function declared in MyInclude.u30sinc
   SendPulseAndWaitAnswer(10, 2s);
}
```

4.3 Constants Declaration

Constants can be declared with the const keyword.

```
const NormalState = StateMachine.Running;
const DefaultTimeout = 450ms;

void main()
{
   WaitForState(
      State     => NormalState,
      Timeout   => DefaultTimeout);
}
```

4.4 Variables Declaration

Variables are instantiated with the var keyword. Variables can be initialized at declaration with a value. If no initial value is specified, the variable will not be initialized.

```
var myVar1;
var myVar2 = 10;
var myVar3 = CounterB;
var myVar4 = myVar1 * myVar2;
```

Unlike with C language, there is no restriction on the location of a variable declaration. Variables can be declared anywhere in the script. The scope of the variable depends on the declaration location.

```
var myGlobalVar = 0;
void MyMacro() { myGlobalVar = 10; }

void main()
{
   var myVar = 10;

   for (var i=0; i<10; i++)
   {
      myVar += 1 << I;
   }

   Sleep ( myVar );
}
```

4.5 Functions Declaration

Functions can be used to save typing and improve the understanding of a script. Functions accept parameters and can optionally return a value.

Optionally, parameters can be prefixed with qualifiers such as in, out or inout. The purpose of these qualifiers is to ensure the user of the function will correctly pass parameters.

The example below shows a script that defines a function for sending a trigger pulse and waiting until an answer is received.

```
void SendPulseAndWaitAnswer(in maxRetries, in maxTime)
{
   repeat(maxRetries)
   {
      GenerateTriggerOut(
         Mode       => PulseHigh);

      WaitTriggerIn(
         Condition => RisingEdge,
         Timeout   => maxTime);

      if(!TimeoutOccured)
      {
         exit;
      }
   }
}

void Main()
{
    SendPulseAndWaitAnswer(10, 2s);
    SendPulseAndWaitAnswer(100, 20ms);
    SendPulseAndWaitAnswer(10, 2s);
}
```

The following example shows a function returning a value based on a parameter:

```
var ComputePosition(index)
{
   return index * 85;
}

TimerA = ComputePosition(CounterB);
```

4.6 Function Calls

The parameters of functions are explicit. The syntax for specifying parameter values is param => value. The parameters order is thus not relevant as the parameter is fully identified by its name. The examples below show a function with two parameters param1 and param2; the value 10 is assigned to param1 and the value 20 to param2.

```
SampleMacro( Param1 => 10, Param2 => 20 );
SampleMacro( Param2 => 20, Param1 => 10 );
```

A function having only one parameter may omit the name of the parameter. For example:

```
Sleep( Duration => 10us );
```

Can also be written as:

```
Sleep( 10us );
```

Parameters are optional when they have a default value. If the parameter is not specified in the call, the default value is used. The example below defines a macro with two parameters. Param1 is mandatory and Param2 has a default value of 0. Since Param2 is not specified in the call, the value 0 will be used as default.

```
void SampleFunction(Param1, Param2 = 0)
{
   Sleep( Param 1 + Param2 );
}

void Main()
{
   SampleFunction( Param1 => 10us );
}
```

4.7 Enumerations Declarations

Enumerations can be used to give names to known values. The example below shows a script that defines several error codes.

```
enum ErrorCode
{
   NoError          = 0,
   Timeout          = 1,
   SequenceMismatch = 2,
   Unspecified      = 3
}
```

The example below shows a script that declares a unique number for each state of a state machine.

```
enum StateMachine
{
   Stopped,
   Paused,
   Running,
   Unspecified
}

void main()
{
   var currentState = GetMachineState();

   if(currentState == StateMachine.Unspecified)
   {
      currentState = StateMachine.Stopped;
   }

   SetMachineState(currentState);
}
```

4.8 Namespace Declarations

Namespaces can be used to isolate some portions of code to avoid name collision in big scripts. The example below shows a script that declares a namespace and then uses functions defined by this namespace.

```
namespace UtilityFunctions
{
   void WaitSpecialEvent(Event, Timeout)
   { /* ... */ }

   void GenerateSpecialEvent(Event, Param = 0)
   {/* ... */ }
}

void WaitAndGenerate(Event)
{
   UtilityFunctions.WaitSpecialEvent(Event, 50ms);
   UtilityFunctions.GenerateSpecialEvent(Event);
}

using UtilityFunctions;

void main()
{
   WaitSpecialEvent(Event, 200ms);
   WaitAndGenerate(Event);
}
```

The example below shows a script that declares two namespaces, each with a function that has the same name.

```
namespace TimingFunctions
{
   void WaitAnswer(Timeout) { /* ... */ }
}

namespace ProtocolFunctions
{
   void WaitAnswer(AnswerId) { /* ... */ }
}

void main()
{
   TimingFunctions.WaitAnswer(400ms);
   ProtocolFunctions.WaitAnswer(Handshake);
}
```

4.9 Inline bytes

Inline data can be specified between square brackets. This inline data can be copied to a memory buffer or used with instructions' parameters. Inline data is the most efficient way of providing data to a parameter because it does not require going through the buffer memory.

```
Buffer[0 to 3] = [ 0, 1, 2, 3 ];

CopyMemory(
   Src         => [ 0x00, 0x00 ],
   Dst         => Buffer[200 for ..]);
```

4.10 Buffers

The hardware contains a buffer of 8192 bytes available for memory comparison and copy operations. It can be accessed with the `Buffer` keyword for reading as well as for writing.

```
Buffer[0 to 3] = [ 0, 1, 2, 3 ];
Buffer[0 for 4] = CounterB;
CounterA = Buffer[10 for 4];
```

The last received packet can be accessed with the `Usb30LastRxPacket` keyword. `Usb30astRxPacket` is read only.

```
Buffer[2 to CounterB] = Usb30LastRxPacket[2 to CounterB];
CounterC = Usb30LastRxPacket[5];
```

4.11 Counters

Counters are useful for example to count errors, special conditions, etc. Several counters are available in the generator, namely `CounterA` to `CounterH`. The value of the counters is indicated in the Registers window.

The example below shows a script that repetitively sends a pulse on the Trigger Out connector and waits for a rising edge on the input Trigger In. If the rising edge is not detected within 500 milliseconds the script increments `CounterA`.

```
repeat(1000)
{
   GenerateTriggerOut(
      Mode        => PulseHigh);

   WaitTriggerIn(
      Condition   => RisingEdge,
      Timeout     => 500ms);

   if(TimeoutOccurred)
   {
      // Keep the error count in CounterA
      CounterA++;
   }
}
```

4.12 Timers

Timers are useful for example to measure or generate precise timing sequences. Several timers are available in the generator. Timers can be started, stopped or modified. It is possible to wait until a timer reaches a specified value or to change the current value of a timer.

The example below shows a script that measure the duration of a trigger pulse and generates one that lasts three times this duration.

```
Timer0 = 0;
Timer1 = 0;

// Wait a rising edge on the input
WaitTriggerIn(Condition => RisingEdge);

// Start Timer0, force trigger output high
StartTimer(0);
GenerateTriggerOut(Mode => ForceHigh);

// Wait a falling edge on the input
WaitTriggerIn(Condition => FallingEdge);

// Stop Timer0. It contains now the duration of the input trigger pulse.
StopTimer(0);

// Start Timer1 with a target value of two times Timer0.
StartTimer(1);
WaitTimer(
   Index          => 1,
   TargetValue    => Timer0 * 2,
   TimingRespect => Hard);

// Force trigger output low
GenerateTriggerOut(Mode => ForceLow);
```

4.13 Ref Keyword

A ref is an abstraction between the Buffer, the `Usb30LastRxPacket` and Inline bytes. It enables writing functions that will accept both Buffer and Inline data without the need of writing the code twice.

When a ref contains inline data all computations must occur at compile time. For example it is possible doing computations by using constants, but it is not possible using runtime registers or counters.

```
void MyOperation(in data, in value)
{
   data[0 for ..] = value;
}

void main()
{
   // Case 1: we take a ref on the Buffer and copy CounterA in it
   ref ref1 = Buffer[0 for 4];
   MyOperation(ref1, CounterA);
   // Case 2: same, but we copy inline data
   ref ref2 = Buffer[0 for 4];
   MyOperation(ref2, [ 1,2,3,4 ] );

   // Case 3: we declare a ref on inline data and copy a constant
   // in it using our function
   ref ref3 = InlineBuffer;
   MyOperation(ref3, [ 1,2,3,4 ] );

   // Case 4: we try the same but with a counter.
   // It will not work because it is not compile time.
   ref ref4 = InlineBuffer;
   MyOperation(ref3, CounterA ); // ERROR: not compile time
}
```

4.14 Stop Keyword

The stop keyword stops the execution of the generator. This is useful for example to stop the generator when a required condition is not met.

```
WaitTriggerIn(
   Condition => FallingEdge,
   Timeout   => 100ms);

if(TimeoutOccurred)
{
   // Condition not met: stop execution
   stop;
}
```

4.15 Breakpoint Keyword

The breakpoint keyword breaks the execution of the generator. The execution can be resumed by the user from the breakpoint.

```
WaitTriggerIn(
   Condition => FallingEdge,
   Timeout   => 100ms);

if(TimeoutOccurred)
{
   // Condition not met: break execution
   breakpoint;
}
```

4.16 If, If Else, and If Else If Statements

The if, if else, and if else if statements executes instructions conditionally depending on a boolean condition. Conditions are described in section 4.23, *Conditional Expressions*.

The example below shows a script incrementing CounterA if the button is pressed, and CounterB otherwise. When CounterA reaches 10, CounterB is reset to 0.

```
WaitButton(
   Index     => 0,
   Timeout   => 0ms,
   Condition => HighLevel);

if(MatchOccurred)
{
   CounterA++;
}
else
{
   CounterB++;
}

if(CounterA >= 10)
{
   CounterB = 0;
}
```

4.17 Switch Statement

The switch statement executes instructions conditionally depending on the value of the specified variable.

The example below shows a script incrementing CounterA if the value of the variable is 0, increments CounterB if the value is 1 and resets both to zero in other cases.

```
switch(CounterC)
{
   case 0:
      CounterA++;
      break;

   case 1:
      CounterB++;
      break;

   default:
      CounterA = 0;
      CounterB = 0;
      break;
}
```

4.18 Repeat Statement

The repeat statement executes instructions the specified count of times. A repeat statement can be stopped with the exit keyword.

The example below shows a script that pulses high the state of the Trigger Out connector for 200 milliseconds every second. It does this 10 times.

```
repeat(10)
{
   GenerateTriggerOut(Mode => ForceHigh);
   Sleep(200ms);
   GenerateTriggerOut(Mode => ForceLow);
   Sleep(800ms);
}
```

4.19 While Statement

The while statement executes instructions as long as a specified condition is true. The condition is checked before the instruction is executed. A while statement can be stopped with the exit keyword.

The example below shows a script that toggles the state of the Trigger Out connector every 200 milliseconds until the Trigger In connector presents a high logic level.

```
while(true)
{
   GenerateTriggerOut(
      Mode      => Toggle);

   WaitTriggerIn(
      Condition => HighLevel,
      Timeout   => 200ms);

   if(MatchOccurre)
   {
      exit;
   }
}
```

4.20 Do While Statement

The do while statement executes instructions as long as a specified condition is true. The condition is checked after the instruction is executed. A while statement can be stopped with the exit keyword.

The example below shows a script that generates a pulse on the Trigger Out connector until the Trigger In connectors presents a high logic level.

```
do
{
   GenerateTriggerOut(
      Mode       => PulseHigh);

   WaitTriggerIn(
      Condition => LowLevel,
      Timeout   => 0);
}
while(MatchOccurred);
```

4.21 For Statement

The for statement executes instructions in a loop a certain number of times. A for statement can be stopped with the exit keyword.

The example below shows a script that generates 20 pulses on the Trigger Out connector.

```
for(var i=0; i<20; i++)
{
   GenerateTriggerOut(Mode => PulseHigh);
}
```

4.22 Mathematical Expressions

The Ellisys script language supports the following mathematical operators:

+, -, *, /, %, &, |, ^, >> and <<.

The examples below show how to use these operators and how to combine them. In all these examples, a must be a variable; b and c can be variables or a literals.

The following example assigns the value 20 to a:

```
a = 20;
```

The following example assigns the value 0xAB12 (43,794 in decimal) to a:

```
a = 0xAB12;
```

The following example adds the value of b to the value of c and assigns the result to a:

```
a = b + c;
```

The following example subtracts the value of c from the value of b and assigns the result to a:

```
a = b - c;
```

The following example multiplies the value of b with the value of c and assigns the result to a:

```
a = b * c;
```

The following example divides the value of b by the value of c and assigns the result to a:

```
a = b / c;
```

The following example divides the value of b with the value of c and assigns the rest of the integer division to a:

```
a = b % c;
```

The following example performs a mathematical AND operation between the value of b and the value of c and assigns the result to a:

```
a = b & c;
```

The following example performs a mathematical OR operation between the value of b and the value of c and assigns the result to a:

```
a = b | c;
```

The following example performs a mathematical XOR operation between the value of b and the value of c and assigns the result to a:

```
a = b ^ c;
```

The following example performs a right shift operation between the value of b and the value of c and assigns the result to a:

```
a = b >> c;
```

The following example performs a left shift operation between the value of b and the value of c and assigns the result to a:

```
a = b << c;
```

The following example demonstrates how to combine expressions to produce more complex results:

```
a = ((b & 0x0F) * 12) >> (c + 1);
```

4.23 Conditional Expressions

The hardware flags that can be tested are `MatchOccurred` and `TimeoutOccured`. These two flags are set by instructions that wait specific conditions.

Conditional expressions can be used as condition of execution or termination with several statements, including `if`, `while` and `do while`.

The following example executes the specified code if a equals b:

```
if(a == b) { /* insert code here */ }
```

The following example executes the specified code if a is different from b:

```
if(a != b) { /* insert code here */ }
```

The following example executes the specified code if a is greater than b:

```
if(a > b) { /* insert code here */ }
```

The following example executes the specified code if a is greater than or equal to b:

```
if(a >= b) { /* insert code here */ }
```

The following example executes the specified code if a is less than b:

```
if(a < b) { /* insert code here */ }
```

The following example executes the specified code if a is less than or equal to b:

```
if(a <= b) { /* insert code here */ }
```

5. Hardware Instructions Set Reference

5.1 Introduction

The Ellisys generator contains a set of instructions implemented in hardware. These hardware instructions are then integrated into higher level functions in order to achieve a complex task. The higher lever functions are then grouped into libraries and supplied in full source code so it is easy to understand the behavior and alter it if needed. Only the hardware instructions are fully documented here, the higher level functions being supplied in source code.

Hardware instructions and functions have exactly the same syntax so it may not be easy to distinguish between both. Hardware instructions are listed in this chapter. All other functions are then just a group of several instructions to achieve a more complex task.

5.2 ConfigureGenerator

The `ConfigureGenerator` instruction applies various commonly used parameters to a script, and must be called at the start of the script. This instruction can be called later in a script in order to change the settings of a given parameter. All parameters in this instruction are optional. If a given parameter is not defined, the value is not changed.

Example

```
ConfigureGenerator(
   in mode                   => GeneratorMode.Host,
   in rxScramblerBypassed    => false,
   in rx8b10bBypassed        => false,
   in rxLanePolarity         => LanePolarity.Normal,
   in txEnableTransceivers   => false,
   in txScramblerBypassed    => false,
   in tx8b10bBypassed        => false,
   in txLanePolarity         => LanePolarity.Normal,
   in txAutoComputeCrcs      => true,
   in txAutoGenerateSkip     => true);
```

Parameter List

Mode	
Description	Specifies the operating mode of the generator.
Range	`GeneratorMode.Disabled` or `GeneratorMode.Host` or `GeneratorMode.Device` or `GeneratorMode.Hub`.
Example	`Disabled` configures the generator to not send traffic. `Host` configures the generator to send traffic on the downstream port. `Device` configures the generator to send traffic on the upstream port. `Hub` configures the generator to send data on both ports.

rxScramblerBypassed	
Description	Specifies whether the traffic received by the generator will be descrambled or not descrambled.
Range	Boolean (True or False).
Example	False will result in descrambling of received traffic. True will result in no descrambling of received traffic.

rx8b10bBypassed	
Description	Specifies whether the traffic received by the generator will go through the 8b/10 decoder or not.
Note	This is an advanced parameter and should only be set to True for specific testing.
Range	Boolean (True or False).
Example	False will result in decoding of received traffic. True will result in no decoding of received traffic.

rxLanePolarity	
Description	Specifies whether the receiver will reverse the polarity of the incoming wires or not.
Range	`LanePolarity.Normal` or `LanePolarity.Inverted`.
Example	`Normal` will result in the receiver not reversing the polarity. `Inverted` will result in the receiver reversing the incoming wires.

txEnableTransceivers	
Description	Specifies whether the transmitter will be enabled or disabled.
Range	Boolean (True or False).
Example	True will enable the transmitter, which will then transmit D0.0 symbols continuously by default, or other symbols when specified. False will disable the transmitter, which will drive the TX lines with electrical idle.

txScramblerBypassed	
Description	Specifies whether the traffic sent by the generator will be scrambled or not scrambled.
Range	Boolean (`True` or `False`).
Example	`False` will result in transmission of scrambled symbols. `True` will result in transmission of non-scrambled symbols.

tx8b10bBypassed	
Description	Specifies whether the traffic sent by the generator will go through the 8b/10b encoder.
Note	This is an advanced parameter and should only be set to True for specific testing.
Range	Boolean (`True` or `False`).
Example	`False` will result in transmission of encoded 10b symbols. `True` will result in transmission of raw 10b symbols.

txLanePolarity	
Description	Specifies whether the traffic sent by the generator will be polarity-reversed or normal polarity.
Range	`LanePolarity.Normal` or `LanePolarity.Inverted`.
Example	`Normal` will result in the transmitter not reversing the polarity. `Inverted` will result in the transmitter reversing the outcoming wires.

txAutoComputeCrcs	
Description	Specifies whether packet CRCs (Header, LCW and Data) will be automatically computed by the generator hardware or sent as specified by the script.
Note	The default auto CRCs computation can be overridden in the `Usb30PushPacket` instruction.
Default	`True`.
Example	`True` will result in the generator automatically computing CRC values. `False` will result in the generator sending the CRC value specified.

txAutoGenerateSkip	
Description	Specifies whether the generator will send Skip ordered sets automatically as required by the specification.
Note	If turned on, skips will be inserted between TSEQ. This parameter should be set to False before sending out TSEQ if it is not the desired behavior.
Range	Boolean (True or False).
Example	True will result in the generator automatically sending Skips. False will result in the generator not sending Skips automatically. Skips can be though inserted manually.

5.3 ConfigureLink

The ConfigureLink instruction configures automated link layer hardware handling to packets received or sent by the generator.

The automatic link layer handling permits the user to greatly simplify the development of complex scripts. All link layers aspects such as link acknowledges (Lgood_N, Lbad), link credits (Lcrd_X), Header Sequence Numbers, etc. are then automatically handled by the generator hardware and the script can focus on higher protocol layers.

Example

```
ConfigureLink(
   in Port                         => DownstreamPort,
   in txAutoHandleLinkAck          => true,
   in txAutoHandleLinkCredit       => true,
   in txAutoComputeHeaderSeqNum    => true);

// Bring link up to U0
// ...

// This will send out automatically the Link Advertisement
ConfigureLink(
   in Port                         => DownstreamPort,
   in rxAutoSendLinkAck            => true,
   in rxAutoSendLinkCredit         => true);
```

Parameters

Port	
Description	Specifies the port to be configured.
Range	UpstreamPort, DownstreamPort, BothPorts
Default	BothPorts

rxAutoSendLinkAck	
Description	Configures the generator to automatically send link acknowledges (Lgood_N, Lbad) on reception of header packets.
Range	`Boolean (True or False)`
Default	`False`

rxAutoSendLinkCredit	
Description	Configures the generator to automatically send link credits (Lcrd_X) on reception of packets.
Note	If turned on, the Lcrd_X will be sent out when the packet is handled in the script by the Usb30WaitPacket instruction.
Range	`Boolean (True or False)`
Default	`False`

rxAutoSendLinkPolling	
Description	Configures the generator to automatically send link pollings (LUP or LDN depending on the destination port) as required.
Range	`Boolean (True or False)`
Default	`False`

txAutoHandleLinkAck	
Description	Configures the generator to automatically handle link acknowledges (Lgood_N, Lbad) on transmission of Header Packets.
Note	If turned on, the generator will wait for the link acknowledge after transmission of a packet with `Usb30PushPacket / Usb30CommitData`. If Lbad is received, the generator will automatically issue a Lretry and retransmit the Header Packet.
Range	`Boolean (True or False)`
Default	`False`

txAutoHandleLinkCredit	
Description	Configures the generator to automatically handle link credits (Lcrd_X) on transmission of Header Packets.
Note	If turned on, the generator will keep track of the Remote Rx Header Buffer Credit. If the script is sending data faster than the remote port can accept, packets transmissions will be paused waiting for new available credits.
Range	Boolean (True or False)
Default	True

txAutoComputeHeaderSeqNumber	
Description	Configures the generator to automatically compute header sequence numbers within the link control word of transmitted header packets.
Note	If turned on, the header sequence number specified in header packets is not relevant and will be overwritten by the generator.
Range	Boolean (True or False)
Default	True.

txAutoHandleItp	
Description	Configures the generator to automatically handle ITPs. These packets will be then no more available in the RX FIFO.
Note	This function is especially useful for upstream ports. The user script is then independent of ITPs and can focus on useful packets.
Range	Boolean (True or False)
Default	False.

txAutoHandleLmp	
Description	Configures the generator to automatically handle LMPs. These packets will be then no more available in the RX FIFO.
Note	This function is especially useful for upstream ports. The user script is then independent of LMPs and can focus on useful packets.
Range	Boolean (True or False)
Default	True.

5.4 Usb30DetectRxTerminations

The `Usb30DetectRxTerminations` instruction determines whether a receiver is present on the far-end of the link with respect to the generator's transmitters, either upstream or downstream.

Example

```
// Wait Remote-Port Terminations
repeat
{
   Usb30DetectRxTerminations();

   if(MatchOccurred)
   {
      // Term detected, proceed to next step
      exit;
   }

   Sleep(6ms);
}

// Next LTSSM steps...
```

Parameter List

Port	
Description	Specifies on which port the Receiver Detection should be done.
Range	`UpstreamPort, DownstreamPort, DefaultPort, BothPorts`
Default	`DefaultPort`
Note	`DefaultPort` will select the port depending on the generator mode. In Device mode the Upstream port will be selected, while in Host mode the downstream port will be selected.

5.5 Usb30PushRawData

The `Usb30PushRawData` instruction commits raw data symbols (any K or D type) into the TX FIFO of the generator. The data can then be committed to be transmitted by using the `Usb30CommitData` instruction.

This instruction can be used to transmit simple ordered sets such as TSEQ, TS1, TS2, SKP, BRST, BERC, BCNT, etc, but also any signature composed of any K or D symbol.

Example

```
Usb30PushRawData(
   Port      => UpstreamPort,
   RawData   => [ COM, COM, COM, COM, 0, 0, D10.2, D10.2, D10.2, D10.2,
                  D10.2, D10.2, D10.2, D10.2, D10.2, D10.2 ],
   Count     => 32);

Usb30CommitData();  // Data will be transmitted after this
```

Parameter List

Port	
Description	Specifies the port on which data will be pushed.
Range	`UpstreamPort`, `DownstreamPort`, `DefaultPort`, `BothPorts`
Default	`DefaultPort`
Example	`DefaultPort` will select the port depending on the generator mode. In Device mode the Upstream port will be selected, while in Host mode the downstream port will be selected.

RawData	
Description	Raw data to be sent.
Type	Inline bytes
Default	No default value; this parameter is mandatory.

Count	
Description	Specifies how many times the specified raw data must be repeated.
Range	`0 to 4,294,967,295`
Default	`1`

5.6 Usb30PushLinkCommand

The `Usb30PushLinkCommand` instruction commits a link command into the TX FIFO of the generator. The link command can then be committed to be transmitted by using the `Usb30CommitData` instruction.

Example

```
Usb30PushLinkCommand(
   Port           => UpstreamPort,
   RawData        => [ 0x01, 0x02 ],
   ComputeCrc     => true);

Usb30CommitData();  // Data will be transmitted after this
```

Parameter List

Port	
Description	Specifies the port on which data will be pushed.
Range	`UpstreamPort, DownstreamPort, DefaultPort, BothPorts`
Default	`DefaultPort`
Example	`DefaultPort` will select the port depending on the generator mode. In Device mode the Upstream port will be selected, while in Host mode the downstream port will be selected.

RawData	
Description	Raw data of the 16-bit link command link command word to be sent.
Note	If the CRC-5 is computed by the generator, the specified CRC-5 value is not relevant and will be overwritten.
Type	Inline bytes or `Buffer`
Default	No default value; this parameter is mandatory.
Example	`[ 0x01, 0x02 ]` to use these bytes for the instruction. `Buffer[0 for 2]` to use bytes from the user buffer.

ComputeCrc	
Description	Specifies if the CRC should be computed automatically by the hardware instead of using the specified value.
Type	Boolean (`True` or `False`)
Default	`True`
Example	`True` to replace the specified CRC bytes with the computed CRC. `False` to leave the specified CRC bytes as is.

5.7 Usb30PushPacket

The `Usb30PushPacket` instruction commits a packet into the TX FIFO of the generator. The packet can then be committed to be transmitted by using the `Usb30CommitData` instruction.

Any kind of packet can be transmitted by using this instruction:

- Link Management Packets (LMP)
- Transaction Packets (TP)
- Data Packets (DP)
- Isochronous Timestamp Packets (ITP)

- But also any invalid or malformed packet

A RawData length of 16 bytes will send a single Header Packet; additional bytes specified will be sent in a succeeding data packet payload (DPP).

Example

```
Usb30PushPacket(
   Port               => UpstreamPort,
   RawData            => [ 0x04, 0x00, 0x00, 0x00, 0x01, 0x00, 0x21, 0x00,
                             0x00, 0x00, 0x00, 0x00, 0xFB, 0x3C, 0x01, 0xE8
],
   ComputeHeaderCrc   => true,
   ComputeLcwCrc      => true,
   ComputeDataCrc     => true);

Usb30CommitData();  // Data will be transmitted after this
```

Parameter List

Port	
Description	Specifies the port on which data will be pushed.
Range	UpstreamPort, DownstreamPort, DefaultPort, BothPorts
Default	DefaultPort
Example	DefaultPort will select the port depending on the generator mode. In Device mode the Upstream port will be selected, while in Host mode the downstream port will be selected.

RawData	
Description	Raw data of the packet to be sent.
Type	Inline bytes or Buffer
Default	No default value; this parameter is mandatory.
Example	[0x01, 0x22, 0xFF] to use these bytes for the instruction. Buffer[0 for 16] to use bytes from the user buffer.

ComputeHeaderCrc	
Description	Specifies if the header CRC-16 should be computed automatically by the hardware instead of using the specified value.
Type	Boolean (True or False)
Default	True
Example	True to replace the specified CRC bytes with the computed CRC. False to leave the specified CRC bytes as is.

ComputeLcwCrc	
Description	Specifies if the link control word CRC-5 should be computed automatically by the hardware instead of using the specified value.
Type	Boolean (True or False)
Default	True
Example	True to replace the specified CRC bytes with the computed CRC. False to leave the specified CRC bytes as is.

ComputeDataCrc	
Description	Specifies if the data packet CRC-32 should be computed automatically by the hardware instead of using the specified value.
Type	Boolean (True or False)
Default	True
Example	True to replace the specified CRC bytes with the computed CRC. False to leave the specified CRC bytes as is.

5.8 Usb30CommitData

The `Usb30CommitData` instruction commits data previously pushed into the TX buffer to be transmitted on the wires.

Example

```
Usb30CommitData();
```

Parameter List

Port	
Description	Specifies the port on which data will be pushed.
Range	`UpstreamPort, DownstreamPort, DefaultPort, BothPorts`
Default	`DefaultPort`
Example	`DefaultPort` will select the port depending on the generator mode. In Device mode the Upstream port will be selected, while in Host mode the downstream port will be selected.

5.9 Usb30WaitOrderedSet

The `Usb30WaitOrderedSet` instruction waits for an ordered set matching the criteria specified.

Example

```
repeat
{
   Usb30WaitOrderedSet(
      Port                          => UpstreamPort,
      WaitOrderedSetTseq            => true,
      WaitOrderedSetTs1             => true,
      WaitOrderedSetTs2             => false,
      WaitOrderedSetBrst            => false,
      WaitOrderedSetBerc            => false,
      WaitOrderedSetBcnt            => false,
      WaitOrderedSetLinkCmd         => false,
      Timeout                       => 500ms);

   if(MatchOccurred)
   {
      // OK we got either a TSEQ or TS1, proceed to next step
      exit;
   }
}
```

Parameter List

Port	
Description	Specifies the port on which data will be waited.
Range	`UpstreamPort, DownstreamPort, DefaultPort`
Default	`DefaultPort`
Example	Default will select the port depending on the generator mode. In Device mode the Upstream port will be selected, while in Host mode the downstream port will be selected.

WaitOrderedSetTseq	
Description	Specifies if a TSEQ ordered set will match.
Type	Boolean (True or False)
Default	True

WaitOrderedSetTs1	
Description	Specifies if a TS1 ordered set will match.
Type	Boolean (True or False)
Default	True

WaitOrderedSetTs2	
Description	Specifies if a TS2 ordered set will match.
Type	Boolean (True or False)
Default	True

WaitOrderedSetBrst	
Description	Specifies if a BRST ordered set will match.
Type	Boolean (True or False)
Default	True

WaitOrderedSetBerc	
Description	Specifies if a BERC ordered set will match.
Type	Boolean (True or False)
Default	True

WaitOrderedSetBcnt	
Description	Specifies if a BCNT ordered set will match.
Type	Boolean (True or False)
Default	True

Timeout	
Description	Timeout after which the instruction is aborted.
Type	Time expressed in seconds.
Range	0 to 34.36 seconds with a precision of 8 nanoseconds.
Default	No default value; this parameter is mandatory.
Example	`1.32ms` means 1,320 microseconds `10ns` will be floored down to 8 nanoseconds.

5.10 Usb30WaitPacket

The `Usb30WaitPacket` Instruction waits for a USB 3.0 packet for further processing.

Example

```
void WaitPacketType(in packetType)
{
   repeat
   {
      Usb30WaitPacket(
         Port     => UpstreamPort,
         Timeout  => 500ms);

      if(MatchOccurred)
      {
         if(Usb30LastRxPacketType == packetType)
         {
            // OK we got a packet of the expected type
            exit;
         }
      }
   }
}
```

Parameter List

Port	
Description	Specifies the port on which data will be waited.
Range	`UpstreamPort, DownstreamPort, DefaultPort`
Default	`DefaultPort`
Example	`DefaultPort` will select the port depending on the generator mode. In Device mode the Upstream port will be selected, while in Host mode the downstream port will be selected.

Timeout	
Description	Timeout after which the instruction is aborted.
Type	Time expressed in seconds.
Range	0 to 34.36 seconds with a precision of 8 nanoseconds.
Default	No default value; this parameter is mandatory.
Example	`1.32ms` means 1,320 microseconds `10ns` will be floored down to 8 nanoseconds.

5.11 HostConfigureBusPowerSource

The `HostConfigureBusPowerSource` instruction configures the bus power source when the generator is configured as host. The generator has the ability to supply the power from an internal source. The voltage of this internal source can then be further configured by using the `HostConfigureInternalVbusLevel` instruction.

Example

```
HostConfigureBusPowerSource( HostPowerSource.Internal );
```

Parameter List

PowerSource	
Description	Selects the power source.
Range	`HostPowerSource.Off` (no power is supplied, Vbus is at 0V) `HostPowerSource.Internal` (power is supplied by the generator) `HostPowerSource.External` (power is supplied from the outside)
Default	`HostPowerSource.Off`

5.12 HostConfigureInternalVbusLevel

The `HostConfigureInternalVbusLevel` instruction configures the voltage supplied by the internal power source. The `HostConfigureBusPowerSource` instruction must be used to select internal power source.

Example

```
HostConfigureBusPowerSource( HostPowerSource.Internal );

var vbusLevel = HostMaxInternalVbusLevel;

while(vbusLevel >= HostMinInternalVbusLevel)
{
   HostConfigureInternalVbusLevel( vbusLevel );
   Sleep(1s);
   vbusLevel -= 020;  // Decrease of 0.20V each iteration
}
```

Parameter List

Level	
Description	Sets the Vbus level.
Type	Value representing 10 mV by unit.
Range	356 (3.56 V) to 543 (5.43 V).
Example	400 means 4.00 V 500 means 5.00 V

5.13 Sleep

The Sleep instruction waits a precise duration which can be specified in units of time (seconds, milliseconds, microseconds and nanoseconds).

Example

```
Sleep( Duration => 1.5ms );
Sleep( 80ns );
```

Parameter List

Duration	
Description	Amount of time to wait.
Type	Time expressed in seconds.
Range	0 to 34.36 seconds with a precision of 8 nanoseconds.
Default	No default value; this parameter is mandatory.
Example	1.32ms means 1,320 microseconds 10ns will be floored down to 8 nanoseconds.

5.14 StartCountdown

The StartCountdown instruction starts a countdown timer in the generator. Three countdown timers can run simultaneously. It is then possible waiting a countdown timer reached zero by using the WaitCountdownReached instruction.

Example

```
StartCountdown ( Index => 0, Duration => 65538us );
StartCountdown ( 65538us );
```

Parameter List

Index	
Description	Index of the countdown timer.
Range	0 to 2.
Default	0

Duration	
Description	Amount of time to wait.
Type	Time expressed in seconds.
Range	0 to 34.36 seconds with a precision of 8 nanoseconds.
Default	No default value; this parameter is mandatory.
Example	1.32ms means 1,320 microseconds 10ns will be floored down to 8 nanoseconds.

5.15 WaitCountdownReached

The WaitCountdownReached instruction waits for the countdown timer to reach zero.

Example

```
WaitCountdownReached(
   Index         => 0,
   Timeout       => 500ms,
   TimingRespect => Hard);
```

Parameter List

Index	
Description	Index of the countdown timer.
Range	0 to 2.
Default	0

Timeout	
Description	Timeout after which the instruction is aborted.
Type	Time expressed in seconds.
Range	0 to 34.36 seconds with a precision of 8 nanoseconds.
Default	No default value; this parameter is mandatory.
Example	1.32ms means 1,320 microseconds 10ns will be floored down to 8 nanoseconds.

TimingRespect	
Description	Specifies if the processor breaks if the countdown value was already reached at the time the wait was called.
Range	Soft or Hard
Default	Soft
Example	Soft to continue even if the countdown value was already reached. Hard to break script execution if the countdown value was exceeded. This value (Hard) helps detecting timing errors in scripts.

5.16 ConfigureTimer

The ConfigureTimer instruction configures specific aspects of the timer instructions.

Example

```
Timer0 = 0;
ConfigureTimer( Index => 0, Prescaler => 1.024ms);
StartTimer( Index => 0 );

const DataTransferSize = 512 * 1024 * 1024;  // 512 MB
DoDataTransfer( DataTransferSize );

StopTimer( Index => 0 );

var readThroughputInKBps = DataTransferSize / Timer0;
```

Parameter List

Index	
Description	Specifies the index of the timer to configure.
Type	`0` to `2`.
Default	No default value; this parameter is mandatory.

Prescaler	
Description	Specifies how many clock counts are needed to increment the timer value. This is helpful for extending the range of a timer if the default range is not enough.
Range	`0` to `4294967295`.
Default	No default value; this parameter is mandatory.
Example	`1` will cause one clock count to increment the specified timer. 10 will result in a timer range of 0 to 343.6 seconds (instead of 34.36) with a precision of 80 ns (instead of 8 ns).

Overflow	
Description	Specifies the value at which the timer will restart at zero.
Range	`0` to `0xFFFFFFFF`.
Default	No default value; this parameter is mandatory.
Example	`000000FF` will use force a restart after 256 clock counts.

5.17 StartTimer

The `StartTimer` instruction starts the specified timer.

Example

```
StartTimer (1);
```

Parameter List

Index	
Description	Specifies the index of the timer to start.
Type	0 to 2.
Default	No default value; this parameter is mandatory.

5.18 StopTimer

The `StopTimer` instruction stops the specified timer.

Example

```
StopTimer (1);
```

Parameter List

Index	
Description	Specifies the index of the timer to stop.
Type	0 to 2.
Default	No default value; this parameter is mandatory.

5.19 WaitTimer

The `WaitTimer` instruction waits until the specified timer reaches the specified value.

Example

```
WaitTimer(
   Index         => 1,
   TargetValue   => 60s);
```

Parameter List

Index	
Description	Specifies the index of the timer to wait on.
Type	0 to 2.
Default	No default value; this parameter is mandatory.

TargetValue	
Description	Specifies the target value to wait on.
Type	0 to 4,294,967,295.
Default	No default value; this parameter is mandatory.
Example	10500 will match when the specified timer reaches value 10,500.

Timeout	
Description	Timeout after which the instruction is aborted.
Type	Time expressed in seconds.
Range	0 to 34.36 seconds with a precision of 8 nanoseconds.
Default	No default value; this parameter is mandatory.
Example	1.32ms means 1,320 microseconds 10ns will be floored down to 8 nanoseconds.

TimingRespect	
Description	Specifies if the processor breaks if the countdown value was already reached at the time the wait was called.
Range	Soft or Hard
Default	Soft
Example	Soft to continue even if the countdown value was already reached. Hard to break script execution if the countdown value was exceeded. This value (Hard) helps detecting timing errors in scripts.

5.20 CopyMemory

The CopyMemory instruction copies bytes from a location of the user buffer to another location.

Example

```
CopyMemory(
   Src         => [ 0x00, 0x00 ],
   Dst         => Buffer[200 for ..]);

CopyMemory(
   Src         => Buffer[0 for 2],
   Dst         => Buffer[200 for ..]);

CopyMemory(
   Src         => Usb30LastRxPacket[16 for 128],
   Dst         => Buffer[0 for ..]);
```

Parameter List

Src	
Description	The source data to copy to the destination
Type	Inline bytes (max 8192 bytes) or `Buffer` or `Usb30LastRxPacket`
Default	No default value; this parameter is mandatory.
Example	`[ 0x00, 0x09, 0x00, 0xE0, 0x00 ]` to copy these bytes. `Buffer[0 for 10]` to copy bytes from the user buffer. `Usb30LastRxBuffer[0 for 16]` to copy the header packet.

Dst	
Description	The destination where the source will be copied.
Type	`Buffer`
Default	No default value; this parameter is mandatory.
Example	`Buffer[100 for ..]` to copy the source to the offset 100 of the user buffer with the length specified in the source.

5.21 WaitTriggerIn

The `WaitTriggerIn` instruction waits on the trigger input SMA connector on the rear of the unit.

Example

```
WaitTriggerIn(
   Condition  => RisingEdge,
   Timeout    => 5s);
```

Parameter List

Condition	
Description	Specifies the trigger condition
Range	`RisingEdge, FallingEdge, HighLevel, LowLevel`
Default	No default value; this parameter is mandatory.
Example	`RisingEdge` waits on a rising edge condition. `HighLevel` waits on a high level (logic high or '1') condition.

Timeout	
Description	Timeout after which the instruction is aborted.
Type	Time expressed in seconds.
Range	0 to 34.36 seconds with a precision of 8 nanoseconds.
Default	No default value; this parameter is mandatory.
Example	1.32ms means 1,320 microseconds 10ns will be floored down to 8 nanoseconds.

5.22 GenerateTriggerOut

The GenerateTriggerOut instruction generates a condition on the Trigger Out SMA connector located on the rear of the unit.

Example

```
GenerateTriggerOut(
   Mode      => PulseHigh);
```

Parameter List

Mode	
Description	Specifies the trigger mode.
Range	PulseHigh, PulseLow, ForceHigh, ForceLow, Toggle
Default	No default value; this parameter is mandatory.
Example	PulseHigh generates a positive pulse on the output. ForceLow forces a low-level (logic low or '0') on the output. Toggle inverts the current level of the output.

6. Special Registers

Special Registers can be used to manage the progression of a script by examining or monitoring the content of these registers.

6.1 Usb30NextTxHeaderSeqNum

This register stores the Header Sequence Number that will be transmitted in the next Header Packet LCW when the Header Sequence Number is computed automatically by the hardware. The `txAutoComputeHeaderSeqNum` parameter of the `ConfigLink` instruction must be set to true for this purpose.

This register is incremented to one for each successfully transmitted packet. This register can be written to reset the Header Sequence Number after a reset, or to create a sequence number violation.

Example

```
ConfigureLink(
   in txAutoComputeHeaderSeqNum    => true);

Usb30NextTxHeaderSeqNum = 0;

Usb30SendPacket(  // HSN = 0
   [ 0x04, 0x00, 0x00, 0x00, 0x01, 0x00, 0x21, 0x00,
     0x00, 0x00, 0x00, 0x00, 0xFB, 0x3C, 0x01, 0xE8 ]);

Usb30SendPacket(  // HSN = 1
   [ 0x04, 0x00, 0x00, 0x00, 0x01, 0x00, 0x21, 0x00,
     0x00, 0x00, 0x00, 0x00, 0xFB, 0x3C, 0x01, 0xE8 ]);

Usb30SendPacket(  // HSN = 2
   [ 0x04, 0x00, 0x00, 0x00, 0x01, 0x00, 0x21, 0x00,
     0x00, 0x00, 0x00, 0x00, 0xFB, 0x3C, 0x01, 0xE8 ]);

Usb30NextTxHeaderSeqNum++;

Usb30SendPacket(  // HSN = 4
   [ 0x04, 0x00, 0x00, 0x00, 0x01, 0x00, 0x21, 0x00,
     0x00, 0x00, 0x00, 0x00, 0xFB, 0x3C, 0x01, 0xE8 ]);
```

6.2 Usb30LastRxPacketType

This register contains the packet type field extracted from the last received packet.

Example

```
void WaitPacketType(in packetType)
{
   repeat
   {
      Usb30WaitPacket(
         Port     => UpstreamPort,
         Timeout  => 500ms);

      if(MatchOccurred)
      {
         if(Usb30LastRxPacketType == packetType)
         {
            // OK we got a packet of the expected type
            exit;
         }
      }
   }
}
```

6.3 Usb30LastRxPacketSubType

This register contains the packet sub-type field extracted from the last received packet. If the field is not applicable for the received packet, then the field is reset to 0.

Example

```
void WaitPacketType(in packetType, in packetSubType)
{
   repeat
   {
      Usb30WaitPacket(
         Port     => UpstreamPort,
         Timeout  => 500ms);

      if(MatchOccurred)
      {
         if(Usb30LastRxPacketType == packetType &&
            Usb30LastRxPacketSubType == packetSubType)
         {
            // OK we got a packet of the expected type and sub-type
            exit;
         }
      }
   }
}
```

6.4 Usb30LastRxPacketDevAddr

This register contains the device address field extracted from the last received packet. If the field is not applicable for the received packet, then the field is reset to 0.

Example

```
void WaitPacketFromAddr(in devAddr)
{
   repeat
   {
      Usb30WaitPacket(
         Port      => UpstreamPort,
         Timeout   => 500ms);

      if(MatchOccurred)
      {
         if(Usb30LastRxPacketDevAddr == devAddr)
         {
            // OK we got a packet from the expected device
            exit;
         }
      }
   }
}
```

6.5 Usb30LastRxPacketEpNum

This register contains the endpoint number field extracted from the last received packet. If the field is not applicable for the received packet, then the field is reset to 0.

Example

```
void WaitPacketFromEndpoint(in devAddr, in epNum)
{
   repeat
   {
      Usb30WaitPacket(
         Port      => UpstreamPort,
         Timeout   => 500ms);

      if(MatchOccurred)
      {
         if(Usb30LastRxPacketDevAddr == devAddr &&
            Usb30LastRxPacketEpNum == epNum)
         {
            // OK we got a packet from the expected endpoint number
            exit;
         }
      }
   }
}
```

6.6 Usb30LastRxPacketNumP

This register contains the NumP (number of packets) field extracted from the last received packet. If the field is not applicable for the received packet, then the field is reset to 0.

6.7 Usb30LastRxPacketPayloadLength

This register contains the length field extracted from the last received packet.

Example

```
WaitPacketType( Usb30PacketType.DataPacketHeader );

// Copy the last RX packet payload to the Buffer at offset 0
Buffer[0 for Usb30LastRxPacketPayloadLength] =
    Usb30LastRxPacket[16 for ..];
```

6.8 Usb30LastRxPacketParams

This register contains some bit fields extracted from the last received packet. This register is a bit field formatted as follow:

- Bit 0: End of Burst
- Bit 1: Endpoint Direction
- Bit 2: Setup TP
- Bit 3: Retry Data Packet

The following constants are defined in the standard Ellisys include files for convenience:

```
const Usb30LastRxPacketParamsEndOfBurst          = 0x00000001;
const Usb30LastRxPacketParamsEndpointDirection   = 0x00000002;
const Usb30LastRxPacketParamsIsSetup             = 0x00000004;
const Usb30LastRxPacketParamsRetryDataPacket     = 0x00000008;
```

6.9 Usb30LastRxPacketErrors

This register contains the errors detected during the last packet reception. This register is a bit field formatted as follow:

- Bit 0: **Header Error**. Any error in the HP only such as:
  a. Not enough symbols (header aborted by another ordered set)
  b. Unexpected symbol (such as K instead of D)
  c. Symbol error (such as 10b/8b decoder error)
  d. Header CRC-16 error
  e. LCW CRC-5 error
- Bit 1: **Packet Error**. Any error in the whole packet including DPP such as:
  f. Any HP error as described above
  g. DPP not back-to-back of HP
  h. Unexpected symbol (such as K instead of D)

- Bit 2: **Type Error**. Type field is unknown.
- Bit 3: **Payload CRC-32 Error**. The CRC-32 of the DPP is incorrect.
- Bit 4: **Payload Length Mismatch**. The length field and the actual DPP received length are not equal.

The following constants are defined in the standard Ellisys include files for convenience:

```
const Usb30LastRxPacketErrorsHeaderError        = 0x00000001;
const Usb30LastRxPacketErrorsPacketError        = 0x00000002;
const Usb30LastRxPacketErrorsPacketTypeError    = 0x00000004;
const Usb30LastRxPacketErrorsDataCrcError       = 0x00000008;
const Usb30LastRxPacketErrorsLengthMismatch     = 0x00000010;
```

Please note that when the link acknowledges are automatically sent by the generator (when the `rxAutoSendLinkAck` of `ConfigureLink` is set to `true`), a packet cannot be received with the Header Error bit set because a Lbad would be sent automatically and the HP would be discarded.

6.10 Usb30NextLinkGoodIndex

This register indicates which index will be used for the next LGOOD. This register is automatically incremented in the following cases:

- The index is incremented for each LGOOD sent after the reception of a valid header packet when Usb30ConfigureLink(rxAutoSendLinkAck) is enabled.
- The index is incremented when sending a LGOOD manually by using the special register Usb30ImmediateLinkCommand.

This register can be written to set the next LGOOD index to be sent at the desired value.

6.11 Usb30NextLinkCreditIndex

This register indicates which index will be used for the next LCRD. This register is automatically incremented in the following cases:

- The index is incremented for each LCRD sent after the handling of a valid header packet when Usb30ConfigureLink(rxAutoSendLinkCredit) is enabled.
- The index is incremented when sending a LCRD manually by using the special register Usb30ImmediateLinkCommand.

This register can be written to set the next LCRD index to be sent at the desired value.

6.12 Usb30ImmediateLinkCommand

This register enables sending link commands with no delay for managing link commands manually. This register is a bit field and writing a one to a bit send out immediately the corresponding link command. The link command is sent in priority independently of the TX FIFO content. This is the main difference with the Usb30PushLinkCommand which goes through the TX FIFO and is thus sent in sequence with the other TX FIFO items.

This register is formatted as follow:

- Bit 0: **Send LGOOD_N**. A LGOOD will be sent with the current index, and the index will be then incremented by one.
- Bit 1: **Send LCRD_X**. A LCRD will be sent with the current credit index, and the credit index will be then incremented by one.
- Bit 2: **Send LDN/LUP**.
- Bit 3: **Send LBAD**.
- Bit 4: **Send LRTY**.

The following constants are defined in the standard Ellisys include files for convenience:

```
const Usb30ImmediateLinkCommandNextLgood        = 0x00000001;
const Usb30ImmediateLinkCommandNextLcredit      = 0x00000002;
const Usb30ImmediateLinkCommandPolling          = 0x00000004;
const Usb30ImmediateLinkCommandBad              = 0x00000008;
const Usb30ImmediateLinkCommandRetry            = 0x00000010;
```

Several bits can be written to 1 at the same time, for example for sending LGOOD/LCREDIT in a unique write.

Example

```
void Usb30LinkAdvertisement()
{
   Usb30NextLinkGoodIndex = 7;
   Usb30NextLinkCreditIndex = 0;
   Usb30ImmediateLinkCommand =  Usb30ImmediateLinkCommandNextLcredit |
                                Usb30ImmediateLinkCommandNextLgood;
   Usb30ImmediateLinkCommand =  Usb30ImmediateLinkCommandNextLcredit;
   Usb30ImmediateLinkCommand =  Usb30ImmediateLinkCommandNextLcredit;
   Usb30ImmediateLinkCommand =  Usb30ImmediateLinkCommandNextLcredit;
}
```

6.13 Usb30OrderedSetCounterTseq

This register contains the count of consecutive TSEQ ordered sets received by the generator.

6.14 Usb30OrderedSetCounterTs1

This register contains the count of consecutive TS1 ordered sets received by the generator.

6.15 Usb30OrderedSetCounterTs2

This register contains the count of consecutive TS2 ordered sets received by the generator.

6.16 Usb30OrderedSetTsLinkFunc

This register contains link functionality field extracted from the last TS1 or TS2 ordered set received by the generator.