Better Analysis.

USB Explorer 260 Generator

User Guide

PROTOCOL TEST SOLUTIONS

This manual is populated throughout with screens captured from a specific version of Ellisys USB Explorer 260 software. All the information contained in the screens are samples and serve as instructional purposes only.

Document Revision History

Date	Revision	Changes
2007-12-12	2.8	Initial release.
2008-04-15	2.8.3000	Added few new instructions (WaitAndSendPacket, HostSetMaxTransactionDuration, HostResetMaxTransactionDuration).

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:	http://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 Product Overview

1.1 Overview

The USB Explorer 260 Generator is a generator for the Universal Serial Bus and protocols based on USB. The USB Explorer 260 Generator verifies product and component reliability by generating reproducible traffic, timing and error scenarios.

The USB Explorer 260 Generator contains a specialized processor specifically designed for the USB protocol. The USB Explorer 260 Generator produces sequences of arbitrary packets with programmable inter-packet delay and can wait for any kind of response packet or event. The processor's instruction set enables you to emulate USB hosts and devices.

1.2 Main Features

The USB Explorer 260 Generator enables you to:

- Emulate most USB equipment, including USB hosts and USB devices.
- Perform functional validation and stress testing of protocol stacks.
- Generate arbitrary packets with full control of the packet raw data down to the PHY layer.
- Generate sequences of packets with programmable inter-packet delay.
- Use exported scripts from protocol analysis software to play back error scenarios.
- Test error recovery mechanisms by generating frames with incorrect content or timing.

Visit the product web page at www.ellisys.com/products/usbex260/ for the latest product information and documentation.

The USB Explorer 260 Generator’s software allows you to quickly and easily create, edit, and debug scripts. Traces previously recorded by an Ellisys USB Explorer 260 Analyzer can be exported to a script and played back by the generator. This will allow you to quickly understand and fix issues that may arise during device, driver or software development.

2 Installing the Ellisys USB Explorer 260

Before installing the Ellisys USB Explorer 260 ensure your computer meets the following requirements:

- Microsoft Windows Installer 3.0 or later. If the installation does not run smoothly, or if the system indicates that there is a version error, update your Windows Installer.
- Microsoft .Net Framework version 2.0.
- Pentium 4, 1.8 GHz or compatible processor, or better.
- 512 MBytes of RAM or more.
- 1024x768 screen display resolution with 256 colors or better.
- USB 2.0 host controller.

2.1 Software Prerequisites

The USB Explorer 260 software requires several software components. Ellisys recommends that you visit the following web pages to update your version of Microsoft .Net Framework and Windows:

- **www.microsoft.com/net** to download the Microsoft .Net Framework version 2.0.
- **windowsupdate.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.

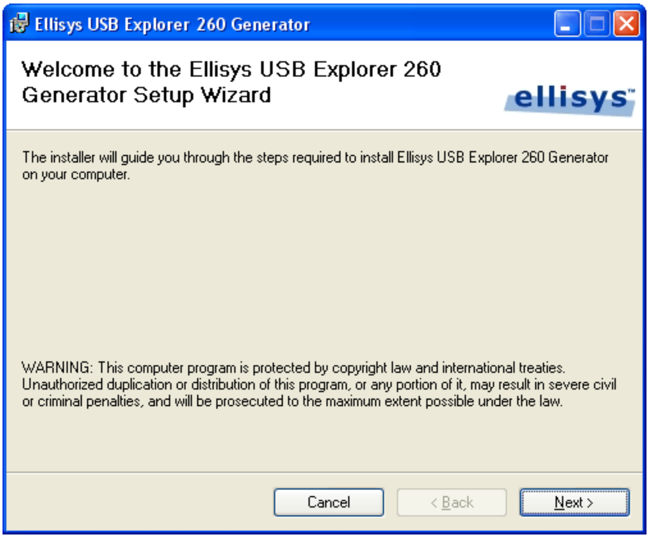
See your System Administrator for more information about updating Microsoft .Net Framework and Windows.

2.2 Installing Software

To install the USB Explorer 260's software:

1. Insert the Ellisys USB Explorer 260 installation CD-ROM that accompanies the product into the computer's CD-ROM drive.

The USB Explorer 260 *Setup Wizard* screen appears:



Ellisys USB Explorer 260 Generator

Welcome to the Ellisys USB Explorer 260 Generator Setup Wizard

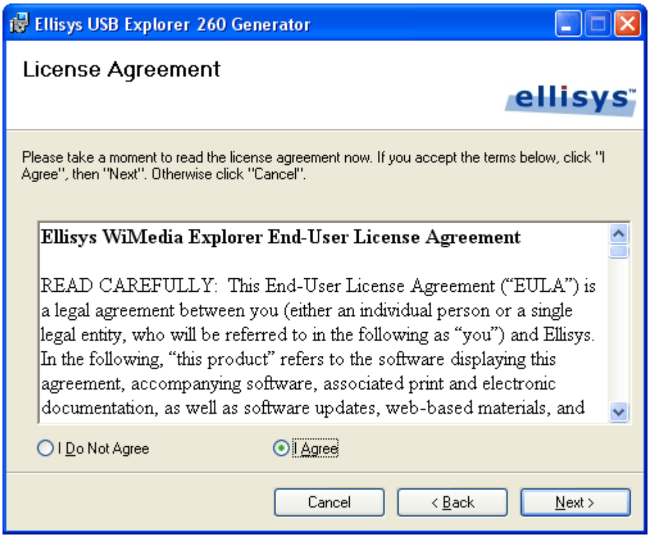
The installer will guide you through the steps required to install Ellisys USB Explorer 260 Generator on your computer.

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 USB Explorer 260 *Setup Wizard* screen does not appear automatically; Click Start | Run, type `d:\setup.exe` (change `d:` to match the drive letter of your CD-ROM) and click on OK.

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

The USB Explorer 260 *Licence Agreement* screen appears:


Ellisys USB Explorer 260 Generator

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 WiMedia Explorer End-User License Agreement

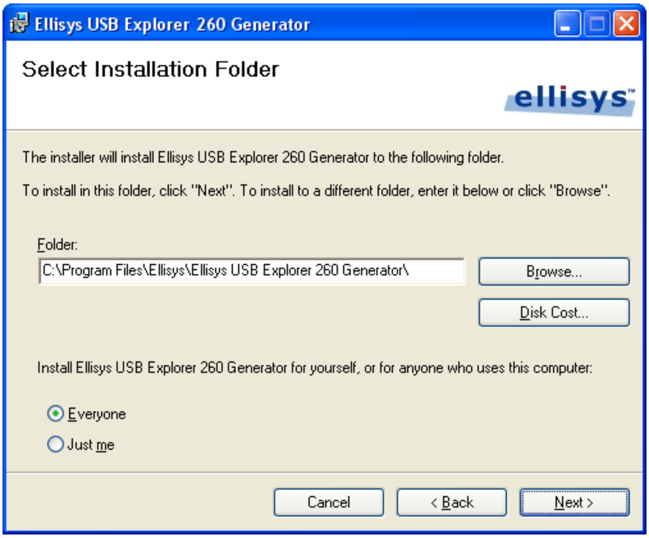
READ CAREFULLY: This End-User License Agreement ("EULA") is a legal agreement between you (either an individual person or a single legal entity, who will be referred to in the following as "you") and Ellisys. In the following, "this product" refers to the software displaying this agreement, accompanying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

I Do Not Agree [✓ I Agree]

3. Read the licence agreement carefully and select **I Agree**.

4. Click on **Next**.

The *Select Installation Folder* screen appears:



Ellisys USB Explorer 260 Generator

Select Installation Folder

The installer will install Ellisys USB Explorer 260 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 260 Generator\

Install Ellisys USB Explorer 260 Generator for yourself, or for anyone who uses this computer:

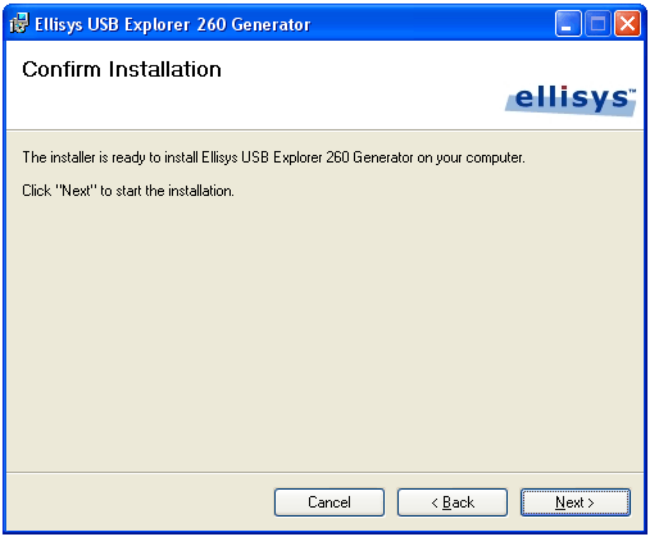
[✓ Everyone]

Just me

5. The default installation folder appears in the *Folder* field. Ellisys recommend 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**.

7. Click on **Next**.

The *Confirm Installation* screen appears:

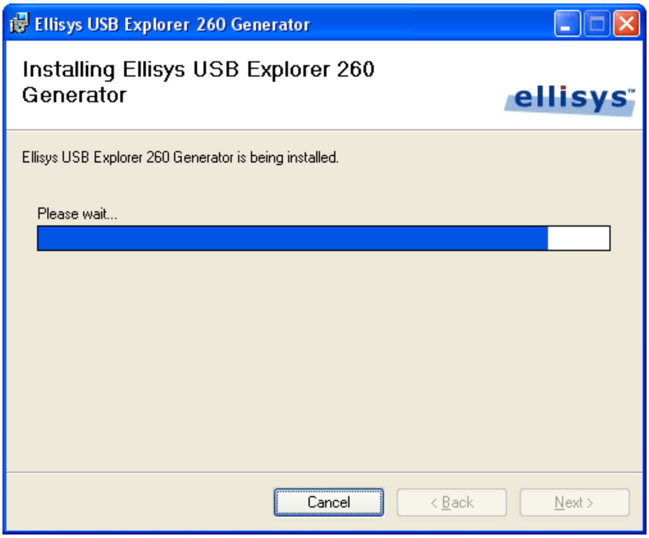


The installer is ready to install Ellisys USB Explorer 260 Generator on your computer.

Click "Next" to start the installation.

8. Click on **Next** to continue the software's installation.

An *Installation Progress* screen appears.

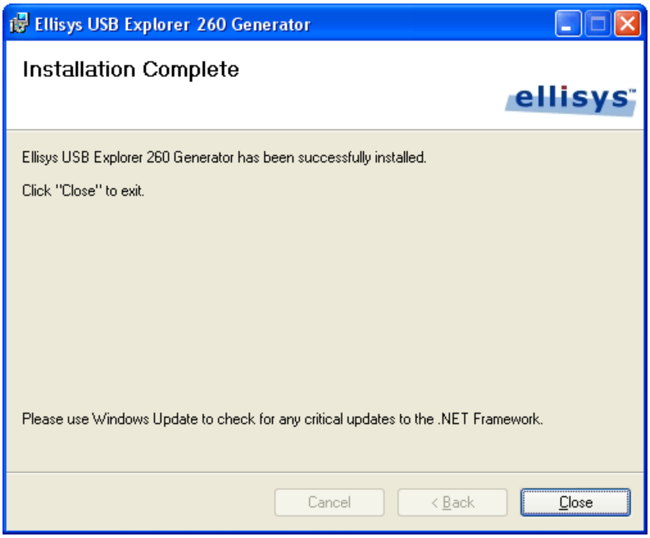

Ellisys USB Explorer 260 Generator

Installing Ellisys USB Explorer 260 Generator

Ellisys USB Explorer 260 Generator is being installed.

Please wait...

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



Ellisys USB Explorer 260 Generator has been successfully installed.

Click "Close" to exit.

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

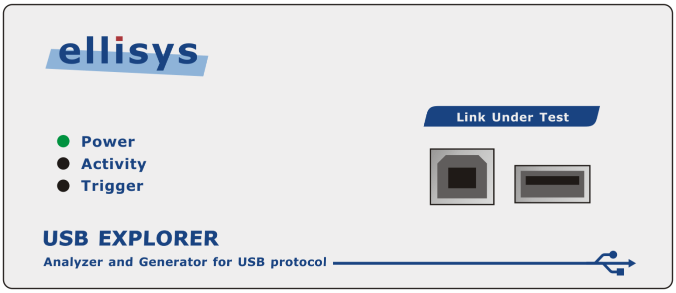
9. Click on **Close**.

The USB Explorer 260 software is now installed.

After installing USB Explorer 260 software a new Hardware Wizard may appear. Refer to *2.5, Connecting to the Computer,* on page 19 for more information about installing the USB driver.

2.3 Front Panel Overview

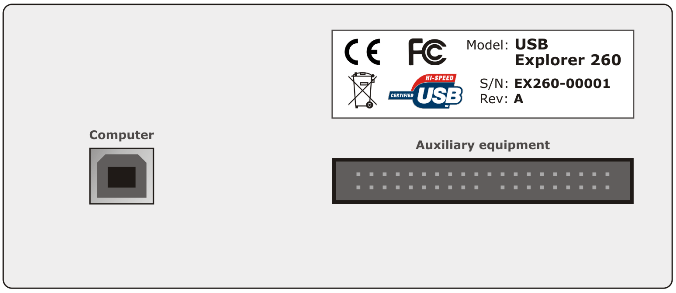
Ellisys USB Explorer 260's front panel:



● **Power**	The *Power* LED is illuminated constant green when connected to a USB 2.0 host controller and working normally.
● **Power**	The *Power* LED is illuminated constant red when connected via a USB 1.1 host controller and working normally. Performance may not be optimal.
● **Power**	The *Power* LED blinks green when connected to a USB 2.0 host controller and the driver is not yet fully installed.
● **Power**	The *Power* LED blinks red when connected to a USB 1.1 host controller and the driver is not yet fully installed.
● **Activity**	The *Activity* LED blinks green when traffic is detected. The blink rate depends on the amount traffic detected, the faster the blink rate the greater amount of traffic detected.
● **Activity**	The *Activity* LED blinks red when traffic is recorded or generated.
● **Trigger**	The *Trigger* LED blinks green when waiting for an event to occur.
● **Trigger**	The *Trigger* LED is illuminated red for a short period when the expected event occurs.

2.4 Back Panel Overview

Ellisys USB Explorer 260's back panel:



A USB cable must be connected between the *Computer* connector and the computer on which the software runs.

When connecting the USB cable <u>DO NOT</u> force the connector into the USB Explorer 260. 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 may break the port connection and is not covered by the warranty.

2.5 Connecting to the Computer

The USB Explorer 260 connects on a USB port, allowing the use of any notebook or desktop computer. The unit is powered by USB and does not require an external adapter. A driver needs to be installed on the computer to ensure proper operation.

Although the USB Explorer 260 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 260, please ensure it is connected on a high speed USB 2.0 enabled host controller before contacting technical support.

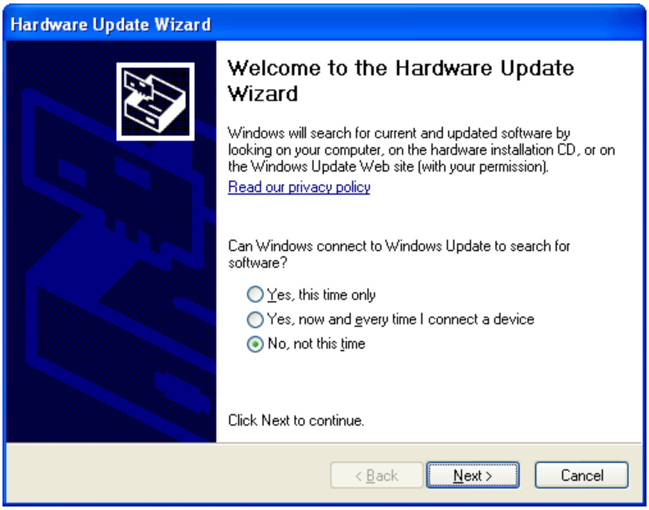
Follow the steps below to install the USB driver:

1. Connect the USB Explorer 260.
   If you are connecting the USB Explorer 260 for the first time wait until Windows displays a message saying a new device has been discovered and go to *Step 3*.

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

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

The *Hardware Update Wizard* window appears:


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?

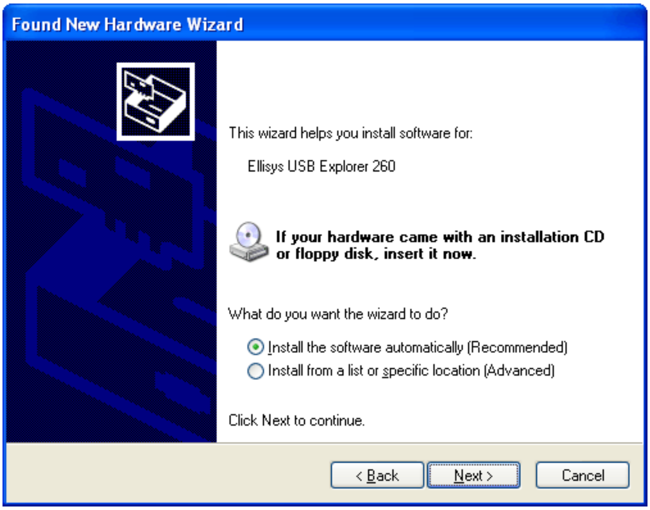
- ( ) Yes, this time only
- ( ) Yes, now and every time I connect a device
- (✓) No, not this time

Click Next to continue.

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

4. Click on **Next**.

The *Found New Hardware* window appears:


Found New Hardware Wizard

This wizard helps you install software for:

Ellisys USB Explorer 260

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

What do you want the wizard to do?

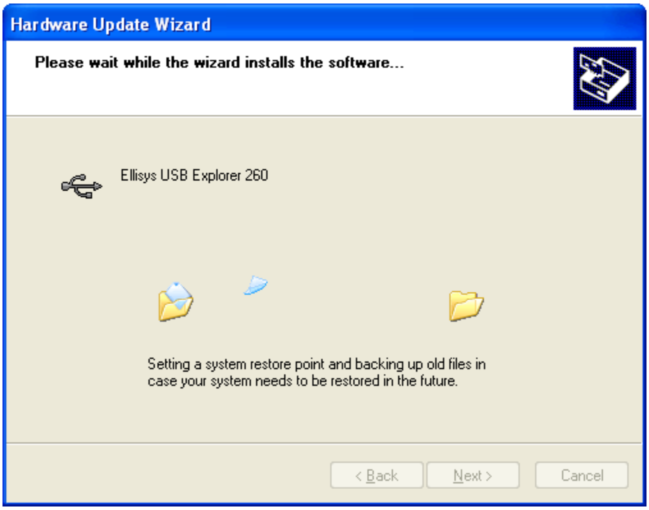
[✓ Install the software automatically (Recommended)]
Install from a list or specific location (Advanced)

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:


Hardware Update Wizard

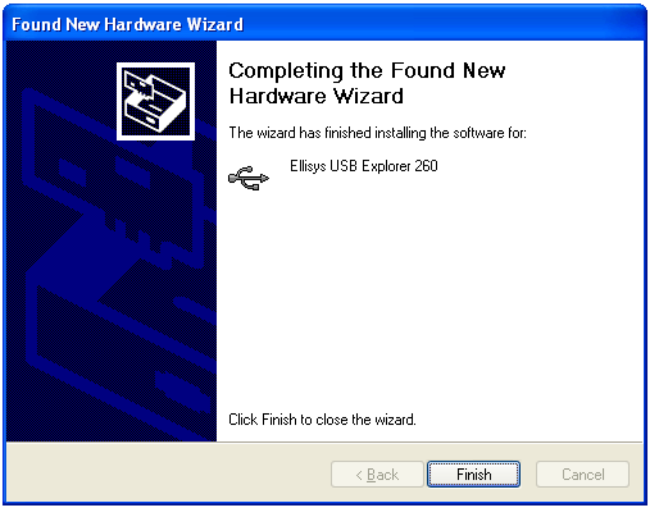
Please wait while the wizard installs the software...

Ellisys USB Explorer 260

Setting a system restore point and backing up old files in case your system needs to be restored in the future.

Windows installs the driver.

When the installation is complete *The wizard has finished installing the software* window appears:



Found New Hardware Wizard

Completing the Found New Hardware Wizard

The wizard has finished installing the software for:

Ellisys USB Explorer 260

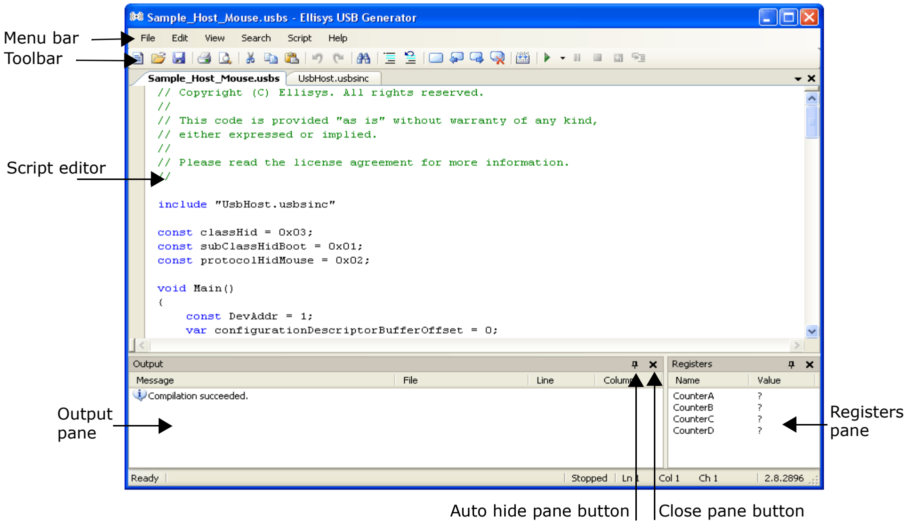
Click Finish to close the wizard.

7. Click on **Finish**.

The installation is complete.

3 User Interface Reference

The user interface of the Ellisys USB Explorer 260 Generator software contains a number of panes, menus, toolbars and other visual elements.



```
// Copyright (C) Ellisys. All rights reserved.
//
// This code is provided "as is" without warranty of any kind,
// either expressed or implied.
//
// Please read the license agreement for more information.
//

include "UsbHost.usbsinc"

const classHid = 0x03;
const subClassHidBoot = 0x01;
const protocolHidMouse = 0x02;

void Main()
{
    const DevAddr = 1;
    var configurationDescriptorBufferOffset = 0;
```

Message	File	Line	Column
Compilation succeeded.			

Name	Value
CounterA	?
CounterB	?
CounterC	?
CounterD	?

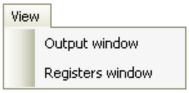
The USB Explorer 260 Generator has several default panes. Each pane displays specific information or allows you to interact with the software for a given task:

- ***Script Editor*** - Shows the current script. The Script Editor also allows editing the script, setting or clearing breakpoints, and placing bookmark to navigate through the script.
- ***Output* pane** - Shows messages about a script after compiling. If there is an error in the script the *Output* pane will show an error description and the error's position: file, line and column.
- ***Register* pane** - Shows the contents of the variables, see *3.14, Working with Registers,* on page 45 for more information.

3.1 Organizing Panes

To open or display a pane:

1. Select **View** in the menu and click on the pane required in the **View** menu.



The selected pane opens.

To close a pane:

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

The pane closes.

To hide a pane:

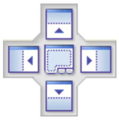
1. Click on **Auto Hide** positioned on the top right-hand corner of the title bar.

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

To move a pane or window:

1. Click on the title bar of a pane or window.
2. Press and hold the left mouse button and drag the pane or window.

A window placer appears:



3. Keep the mouse button pressed 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 260 Generator toolbar buttons and their actions.

	New Document	Opens a new document.
	Open Document	Opens a folder to allow you open a previous saved file.
	Save Document	Saves a document.
	Print	Opens print options to allow you to print a document.
	Print Preview	Opens the print preview window.
	Cut	Cuts a selection of text.
	Copy	Copies a selection of text.
	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	Uncomment 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 script when running.
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 USB Explorer 260 Generator main menu options and their actions.

File

	New (CTRL + N)	Creates a new file.
	Open (CTRL + O)	Opens a previous saved file.
	Save (CTRL + S)	Saves a file.
	Save As	Saves a file with a new name.
	Page Setup	Opens the Page Setup dialog box that lets you set the page margins and other parameters.
	Print Preview	Opens the Print Preview window.
	Print (CTRL + P)	Prints a file.
	Exit	Exits the software.

Edit

	Undo (CTRL + Z)	Undoes the previous action.
	Redo (CTRL + Y)	Redoes the previous action.
	Cut (CTRL + X)	Cuts a selection of text.
	Copy (CTRL + C)	Copies a selection of text.

	Paste (CTRL + V)	Pastes a selection of copied or cut text.

Edit | Advanced

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

Edit | Bookmarks

	Toggle Bookmark	Toggles a bookmark at a selected line.
	Enable Bookmark	Enables the selected bookmark.
	Previous Bookmark	Finds the previous bookmark.
	Next Bookmark	Finds the next bookmark.
	Clear Bookmarks	Clears all bookmarks.
	Insert Snippet Code (CTRL + I)	Opens the Insert Snippet code list.

View

	Output window	Opens or closes the Output window.
	Registers window	Opens or closes the Registers window.

Search

	Find/Replace (CTRL + F)	Opens the Find/Replace window.
	Find Next F3	Finds the text previously entered in the Find/Replace window.
	Find Previous (SHIFT + F3)	Finds the text previously entered in the Find/Replace 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** (SHIFT +F5)	Stops a running script.
	Restart	Stops and restarts a script from the beginning.
	Step (F10)	Steps from line to line in the script.
●	**Toggle Breakpoint** (F9)	Toggles a breakpoint at a selected line.

	Clear all Breakpoints (CTRL+SHIFT +F9)	Removes all breakpoints in the script.
	Select a Generator	Opens the Available Generators window.

Help

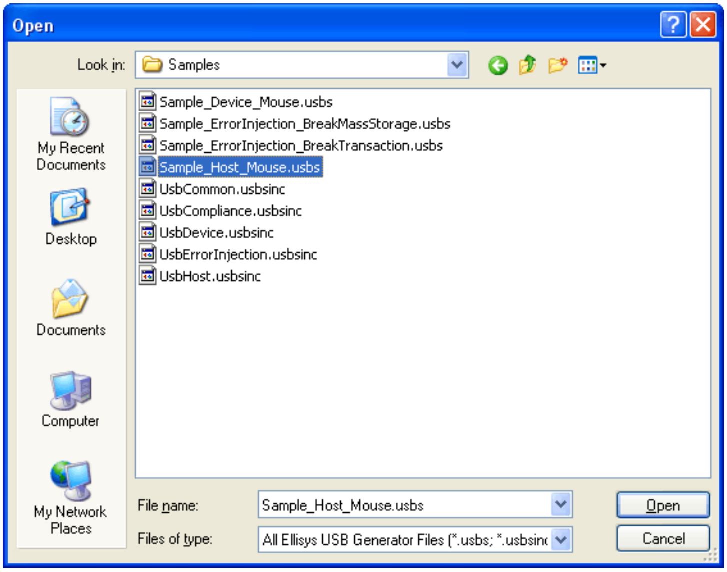
	User Guide	Opens the online user guide.
	Ellisys website	Opens the Ellisys website in your default internet browser.
	Contact support	Opens a form to contact the technical support.
	About	Opens the About window.

3.4 Opening a File

To open a file:

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

The *Open File* window appears:



Sample_Device_Mouse.usbs
Sample_ErrorInjection_BreakMassStorage.usbs
Sample_ErrorInjection_BreakTransaction.usbs
Sample_Host_Mouse.usbs
UsbCommon.usbsinc
UsbCompliance.usbsinc
UsbDevice.usbsinc
UsbErrorInjection.usbsinc
UsbHost.usbsinc

File name: Sample_Host_Mouse.usbs
Files of type: All Ellisys USB Generator Files (*.usbs; *.usbsinc

2. Select the file required and click on **Open**.

The selected file opens in the software.

3.5 Saving a File

To save a file:

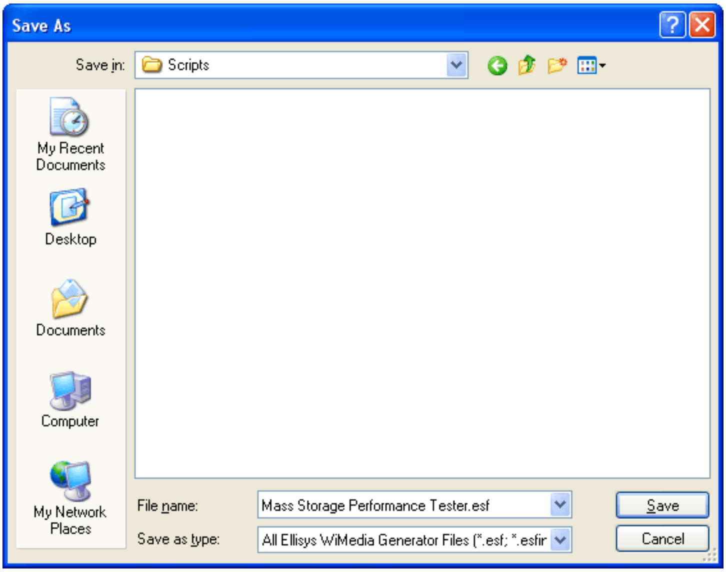
1. Select **File | Save** in the menu or click on **Save Document** .

The file is saved.

To save a file with a new name:

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

The *Save As* window appears:


Save As
Save in: Scripts
My Recent Documents
Desktop
Documents
Computer
My Network Places
File name: Mass Storage Performance Tester.esf
Save as type: All Ellisys WiMedia Generator Files (*.esf; *.esfir
Save
Cancel

2. Navigate to the directory where the file is to be saved.
3. Enter the required name of the file in the *File name* field and click on **Save**.

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

3.6 Printing a File

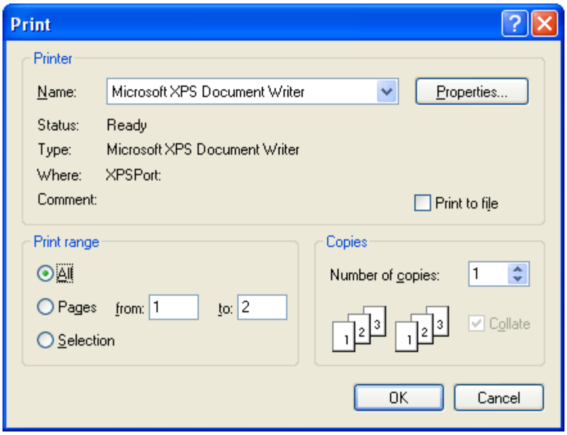
Use the Page Setup option, **File | Page Setup**, to setup how the file should be printed. This option will depend on the printer, please see your printer's documentation for more information.

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:


Print

Printer

Name: Microsoft XPS Document Writer

Status: Ready

Type: Microsoft XPS Document Writer

Where: XPSPort:

Comment:

Print to file [ ]

Print range

[✓ All]

Pages from: 1 to: 2

Selection

Copies

Number of copies: 1

Collate [✓]

2. Select the printer and printer setup if required.
3. Click on **OK**.

The file is printed.

3.7 Editing a Script

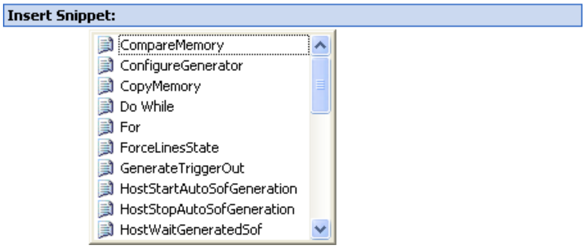
The USB Explorer 260 Generator includes several specialized instructions. Example code for these instructions can be inserted to help you write instructions. An example code is called a code snippet.

A full description of the specialized instructions can be found in *Chapter 5, Instruction Set Reference,* on page 63.

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 code snippet required from the list.

4. Double-click on the code snippet required.
or
Select the snippet required and press ENTER.

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

3.8 Advanced Editing Features

All the USB Explorer 260 Generator's advanced editing features 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 mark beside the line.

To highlight the current line:

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

The line with the cursor is highlighted.

To display the column 80 guide:

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

The 80 column guide appears as a line in the main script pane.

To comment a selection in a script:

1. Select the lines you want to comment.

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 a script:

1. Select the commented lines you want to uncomment.

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 text required in the script.
2. Select **Edit | Advanced | Make Uppercase** to change the text's case from lowercase to uppercase.

or

3. Select **Edit | Advanced | Make Lowercase** to change the text's case from uppercase to lowercase.

3.9 Searching

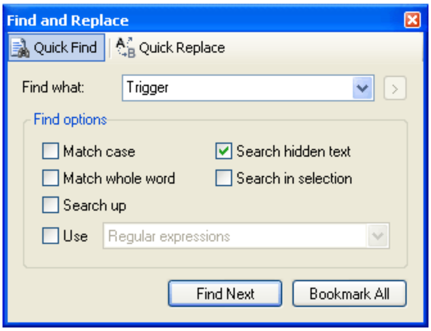
Search, find and replace options can be accessed by clicking **Search** in the menu.

To search text:

1. Click on **Find/Replace**

   or
   Select **Search | Find** in the menu.
   or
   Press CTRL + F.

The *Find/Replace* window appears:


Find and Replace

Quick Find | Quick Replace

Find what: Trigger

Find options

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

Find Next | Bookmark All

2. Enter what you need to be found in the *Find what* field.

or

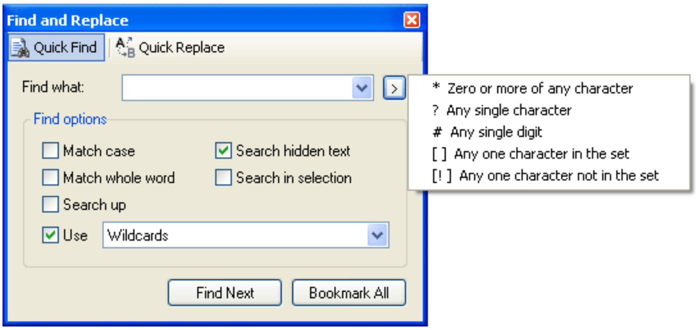
3. Select the **Use** check box if you want to use Regular expression or Wildcards.

Regular expressions or Wildcards can be selected as an option.

4. If you selected the **Use** check box, select *Regular expression* or *Wildcards* from the drop-down list. The **Right Arrow** > beside the *Find What* field becomes enabled.

5. Click on **Right Arrow** > .

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



Find and Replace

Quick Find | Quick Replace

Find what:

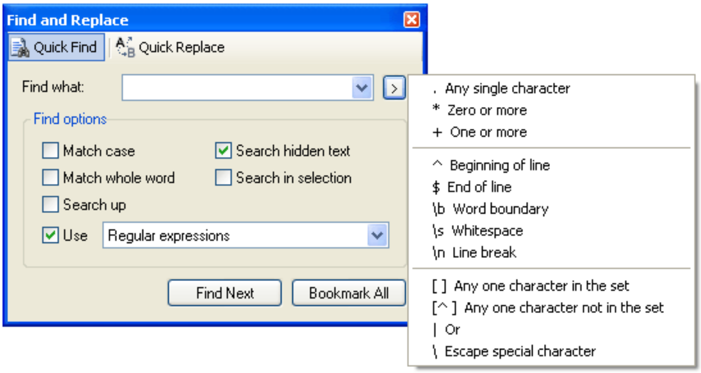
Find options

- [ ] 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

6. Select the Wildcard required.

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



Find and Replace

Quick Find | Quick Replace

Find what:

Find options

- [ ] 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

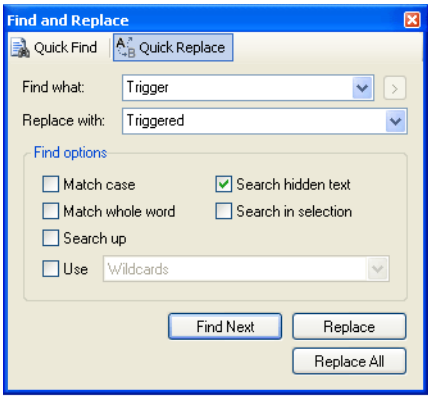
7. Select the Regular expression required.
8. Select the required search options check boxes.
9. Click on the required button: **Find Next** to find the next occurrence or **Bookmark All** to bookmark all occurrences.

The selected search is performed.

To replace text:

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

The *Find/Replace* window 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 what you need to be found in the *Find what* field.
3. Enter the replacement text in the *Replace with* field.
4. Select the required search options check boxes.
5. Click on the required button: **Find Next** to find the next occurrence or **Replace or Replace All** to respectively replace the next occurrence or all occurrences.

The selected replacement is performed.

3.10 Working with Bookmarks

A bookmark is a useful tool that enables you to mark lines of code to help you navigate through a script.

All the 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.

2. 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.
2. 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.

2. 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 Bookmark**
   or
   Select **Edit | Bookmarks | Clear Bookmark** in the menu.

All bookmarks in the script are removed.

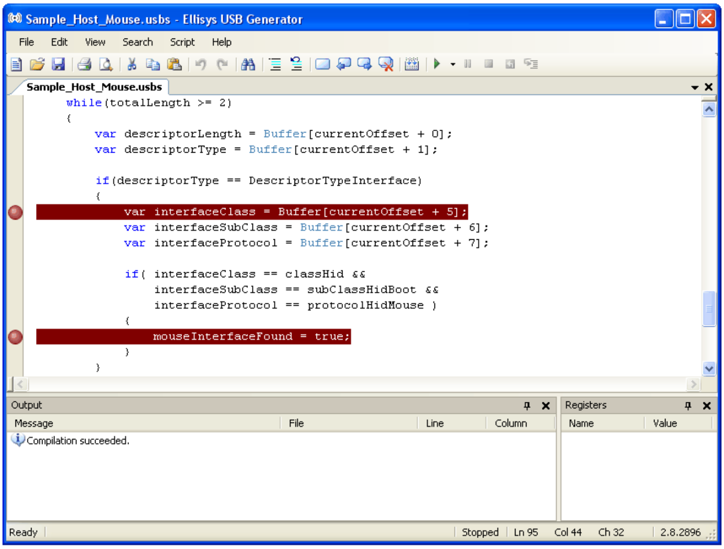
3.11 Working with Breakpoints

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_Host_Mouse.usbs - Ellisys USB Generator
File Edit View Search Script Help
Sample_Host_Mouse.usbs
Output
Message File Line Column
Registers
Name Value
Ready Stopped Ln 95 Col 44 Ch 32 2.8.2896

```
    while(totalLength >= 2)
    {
        var descriptorLength = Buffer[currentOffset + 0];
        var descriptorType = Buffer[currentOffset + 1];

        if(descriptorType == DescriptorTypeInterface)
        {
            var interfaceClass = Buffer[currentOffset + 5];
            var interfaceSubClass = Buffer[currentOffset + 6];
            var interfaceProtocol = Buffer[currentOffset + 7];

            if( interfaceClass == classHid &&
                interfaceSubClass == subClassHidBoot &&
                interfaceProtocol == protocolHidMouse )
            {
                mouseInterfaceFound = true;
            }
        }
```

```
Compilation succeeded.
```

To remove all breakpoints:

1. Select **Script | Clear All Breakpoint** in the menu.

All breakpoints in the script are removed.

3.12 Compiling a Script

To compile a script:

1. Open a script file as described in *3.4, Opening a File,* on page 33.
   or
   Create a new script file and save it.

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

The USB Explorer 260 Generator compiles the script.

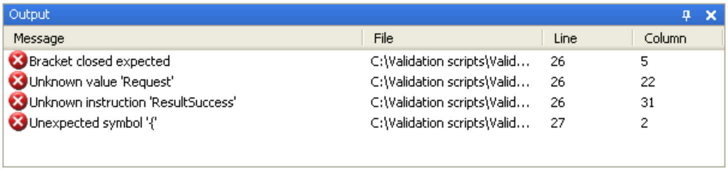
If the compilation is successful a *'Compilation Succeeded'* message will appear in the *Output* pane.

If the compilation is unsuccessful a *'Compilation Failed'* message will appear in the *Output* pane. A list of errors will also be listed in the *Output* pane.

To find an error in a compiled script:

1. Compile a script as described in *3.12, Compiling a Script,* on page 42.

The compilation errors are listed in the *Output* pane under the *Message* column.



Output

Message	File	Line	Column
Bracket closed expected	C:\Validation scripts\Valid…	26	5
Unknown value 'Request'	C:\Validation scripts\Valid…	26	22
Unknown instruction 'ResultSuccess'	C:\Validation scripts\Valid…	26	31
Unexpected symbol '{'	C:\Validation scripts\Valid…	27	2

2. Double-click on the error description you require in the *Output* pane.

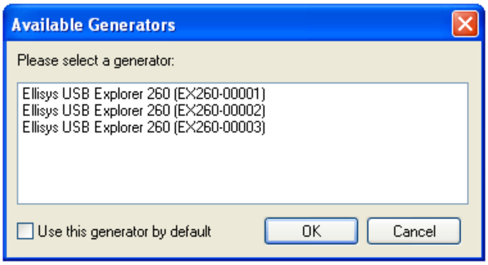
The line that contains the error is highlighted in the main script pane.

3.13 Running a Script

To select a generator:

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

The *Available Generators* window appears:



Available Generators

Please select a generator:

Ellisys USB Explorer 260 (EX260-00001)
Ellisys USB Explorer 260 (EX260-00002)
Ellisys USB Explorer 260 (EX260-00003)

2. Select the required generator and click on **OK**.

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

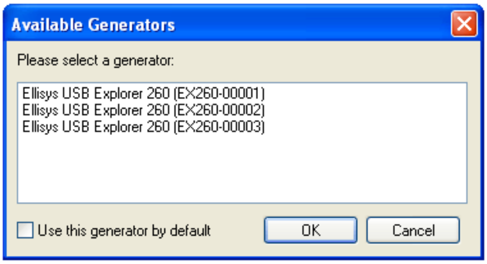
The generator is selected.

To run a script:

1. Open a script file as described in *3.4, Opening a File,* on page 33
   or
   Create a new script file and save it.

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

If you did not select a generator as a default generator then the *Available Generators* window appears:


Available Generators

Please select a generator:

Ellisys USB Explorer 260 (EX260-00001)
Ellisys USB Explorer 260 (EX260-00002)
Ellisys USB Explorer 260 (EX260-00003)

3. Select on the required generator and click on **OK**.

The script runs using the selected generator.

To break or pause a script:

1. Run a script as described in *3.13, Running a Script,* on page 43.

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

The script is paused.

To stop a script:

1. Run a script as described in *3.13, Running a Script,* on page 43.
2. Click on **Stop**
   or
   Select **Script | Stop** in the menu.

The script stops.

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

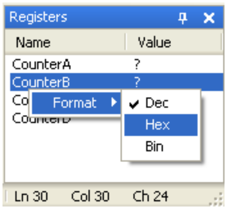
This section describes how you can work with registers. For more information about registers see *4.10, Counters,* on page 53.

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 format require; **Dec**, **Hex** or **Bin**.

The register format is changed to the selected format and the numbers are displayed.

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 open using the `/*` characters, and are closed using the `*/` characters.

```
void Main()
{
/* This is a multi line comment the prevents the
following instruction to be 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 includes a file and use then the macro declared inside.

```
include "MyInclude.esf"

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

4.3 Constants Declaration

Constants can be declared with the `const` keywork.

The example below shows a script that defines two constants.

```
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. The variable can be initialized at declaration with a value. If no initial value is specified the variable will not be initialized.

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

There is no restriction on variables declaration location. 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 = 0;

        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.

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

```
void SendPulseAndWaitAnswer(MaxRetries,
                            MaxTime)
{
    repeat(MaxRetries)
    {
        GenerateTriggerOut(Output => BncOut,
                           Mode   => PulseHigh);

        WaitTriggerIn(Input     => BncIn,
                      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 ComputeSlotPosition(Index)
{
       return Index * 85;

}


TimerA = ComputeSlotPosition(CounterB);
```

4.6 Function Calls

The parameters of functions calls are explicit. The syntax for specifying parameters values is `param => value`. The parameters order is thus not relevant as the parameter is fully identified by its name. The examples below shows a function with two parameters `Param1` and `Param2`; the value 10 is assigned to `Param1` and the value 20 is assigned to `Param2`:

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

When an instruction, a function has only one parameter its name can be omitted. 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 Namespaces 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 use 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 scripts that declare 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 Buffer Usage

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. Example:

```
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 `LastRxPacket` keyword. `LastRxPacket` is read only. Example:

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

4.10 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 output BNC connector and waits for a rising edge on the input BNC connector. If the rising edge is not detected within 500 milliseconds the script increments `CounterA`.

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

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

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

4.11 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;

WaitTriggerIn(Input     => BncIn,
              Condition => RisingEdge);

StartTimer(0);

GenerateTriggerOut(Output => BncOut,
                   Mode   => ForceHigh);

WaitTriggerIn(Input     => BncIn,
              Condition => FallingEdge);

StartTimer(1);
StopTimer(0);

WaitTimer(Index         => 1,
          TargetValue   => Timer0 * 2,
          TimingRespect => Hard);

GenerateTriggerOut(Output => BncOut,
                   Mode   => ForceLow);

StopTimer(1);
```

4.12 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(Input     => BncIn,
              Condition => FallingEdge,
              Timeout   => 100ms);

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

4.13 Breakpoint Keyword

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

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

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

4.14 If Statement

The `if` statement executes instructions conditionally depending on a condition. Conditions are described in *4.21, Conditional expressions,* on page 61.

The example below shows a script that increments `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;
}
```

4.15 Switch Statement

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

The example below shows a script that increments `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;
}
```

4.16 Repeat Statement

The `repeat` statement executes instructions the specified count of times. A repeat statement can be stopped with the exit keyword. Up to four repeat statements can be imbricated.

The example below shows a script that pulses high the state of the output BNC connector for 200 milliseconds every seconds. It does this 10 times.

```
repeat(10)
{
    GenerateTriggerOut(Output => BncOut,
                       Mode   => ForceHigh);

    Sleep(200ms);

    GenerateTriggerOut(Output => BncOut,
                       Mode   => ForceLow);

    Sleep(800ms);
}
```

4.17 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. Up to four while statements can be imbricated.

The example below shows a script that toggles the state of the output BNC connector every 200 milliseconds until the input BNC connector presents a high logic level.

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

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

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

4.18 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. Up to four do while statements can be imbricated.

The example below shows a script that generates a pulse on the output BNC connector until the input BNC connectors presents a high logic level.

```
do
{
    GenerateTriggerOut(Output => BncOut,
                       Mode   => PulseHigh);

    WaitTriggerIn(Input     => BncIn,
                  Condition => LowLevel,
                  Timeout   => 0);

} while(MatchOccurred);
```

4.19 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. Up to four for statements can be imbricated.

The example below shows a script that generates 20 pulses on the output BNC connector.

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

4.20 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 subtract 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);
```

4.21 Conditional expressions

The conditions 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 Instruction Set Reference

The Ellisys USB Explorer 260 Generator includes several specialized instructions. These instructions are divided into six distinct categories:

- Timing operations
- Buffer operations
- Trigger operations
- Link-oriented operations
- Packet-oriented operations
- Host-oriented operations

5.1 Sleep Instruction

The `Sleep` instruction waits a precise duration which can be specified in several units. The duration can be specified in units of time (seconds, milliseconds, microseconds and nanoseconds) or in 60 MHz clock cycles.

Example

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

Parameter List

Duration	
Description	Amount of time to wait.
Type	Time expressed in 60 MHz clock cycles or seconds.
Range	0 to 4,294,967,295 clock cycles or 0 to 71 seconds with a precision of 16.$\overline{6}$6 nanoseconds.
Default	No default value; this parameter is mandatory.
Example	**`1.32ms`** means 1,320 microseconds or 79,200 clock cycles. **`620ns`** will be floored down to 37 clock cycles. **`3960clk`** means 3,960 clock cycles or 66 microseconds. `1000` (without unity) is not allowed and will generate a warning.

5.2 StartCountdown Instruction

The `StartCountdown` instruction starts a countdown timer in the generator. Two countdown timers can run simultaneously.

Example

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

Parameter List

Index	
Description	Index of the countdown timer.
Range	0 to 2.
Default	`0`
Example	**`0`** to use the countdown timer with index 0. **`1`** to use the countdown timer with index 1.

Duration	
Description	Amount of time to wait.
Type	Time expressed in 60 MHz clock cycles or seconds.
Range	0 to 4,294,967,295 clock cycles or 0 to 71 seconds with a precision of 16.6$\overline{6}$ nanoseconds.
Default	No default value; this parameter is mandatory.
Example	**`1.32ms`** means 1,320 microseconds or 79,200 clock cycles. **`620ns`** will be floored down to 37 clock cycles. **`3960clk`** means 3,960 clock cycles or 66 microseconds. **`1000`** (without unity) is not allowed and will generate a warning.

5.3 WaitCountdownReached Instruction

The `WaitCountdownReached` instruction waits the countdown timer reaches its nominal value.

Example

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

Parameter List

Index	
Description	Index of the countdown timer.
Range	0 to 2.
Default	0
Example	`0` to use the countdown timer with index 0. `1` to use the countdown timer with index 1.

Timeout	
Description	Timeout after which the instruction is aborted.
Type	Time expressed in 60 MHz clock cycles or seconds.
Range	0 to 4,294,967,295 clock cycles or 0 to 71 seconds with a precision of $16.\overline{6}6$ nanoseconds.
Default	No default value; this parameter is mandatory.
Example	**`1.32ms`** means 1,320 microseconds or 79,200 clock cycles. **`620ns`** will be floored down to 37 clock cycles. **`3960clk`** means 3,960 clock cycles or 66 microseconds. **`1000`** (without unity) is not allowed and will generate a warning.

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 helps detecting timing errors in scripts.

5.4 StartTimer Instruction

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.
Example	**`0`** will use timer 0.

5.5 StopTimer Instruction

The `StopTimer` instruction stops the specified timer.

Example

```
StopTimer(2);
```

Parameter List

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

5.6 WaitTimer Instruction

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.
Example	`0` will use timer 0.

TargetValue	
Description	Specifies the target value to wait on.
Type	0 to 4,294,967,295 clock cycles or 0 to 71 seconds with a precision of 16.6$\overline{6}$ nanoseconds.
Default	No default value; this parameter is mandatory.
Example	`10500` will match when the specified timer reaches value 10500. `200ms` will match when the specified timer reaches value 12,000,000, which equals to 200ms at 60 MHz.

Timeout	
Description	Timeout after which the instruction is aborted.
Type	Time expressed in 60 MHz clock cycles or seconds.
Range	0 to 4,294,967,295 clock cycles or 0 to 71 seconds with a precision of 16.6$\overline{6}$ nanoseconds.
Default	No default value; this parameter is mandatory.
Example	`1.32ms` means 1,320 microseconds or 79,200 clock cycles. `620ns` will be floored down to 37 clock cycles. `3960clk` means 3,960 clock cycles or 66 microseconds. `1000` (without unity) is not allowed and will generate a warning.

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 helps detecting timing errors in scripts.

5.7 CopyMemory Instruction

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

Example

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

CopyMemory(
        Src                 => Buffer,
        SrcOffset           => 0,
        Dst                 => Buffer,
        DstOffset           => 200,
        Length              => 2);

CopyMemory(
        Src                 => LastRxPacket,
        SrcOffset           => 15,
        Dst                 => Buffer,
        DstOffset           => 15,
        Length              => 60);
```

Parameter List

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

SrcOffset	
Description	Offset in the source data of the first byte to use.
Range	0 to 8191.
Default	`0`
Example	**`0`** will copy from the beginning of the PHY header when `LastRxPacket` is specified or from offset 0 of the user buffer when `Buffer` is used. **`5`** will copy from the beginning of the MAC header when `LastRxPacket` is specified or from offset 5 of the user buffer when `Buffer` is used. **`15`** will copy from the beginning of the payload when `LastRxPacket` is specified or from offset 15 of the user buffer when `Buffer` is used.

Dst	
Description	The destination where the source will be copied.
Type	`Buffer`
Default	No default value; this parameter is mandatory.
Example	**`Buffer`** is the only acceptable value.

DstOffset	
Description	Offset in the destination buffer of the first data byte to copy.
Range	0 to 8191.
Default	No default value; this parameter is mandatory.
Example	**`0`** will copy source bytes at offset 0 of the destination buffer. **`22`** will copy source bytes at offset 22 of the destination buffer.

Length	
Description	Length of the data to copy.
Range	0 to 8192.
Default	No default value; this parameter is mandatory.
Example	**`5`** will copy 5 bytes.

5.8 CompareMemory Instruction

The `CompareMemory` instruction compares bytes from a location of the user buffer to another.

Example

```
CompareMemory(
        Src                 => Buffer,
        SrcOffset           => 60,
        Dst                 => [ 0x00, 0x00 ] );

CompareMemory(
        Src                 => Buffer,
        SrcOffset           => 0,
        Dst                 => Buffer,
        DstOffset           => 200,
        Length              => 40);

CompareMemory(
        Src                 => LastRxPacket,
        SrcOffset           => 5,
        Dst                 => Buffer,
        DstOffset           => 5,
        Length              => 10);
```

Parameter List

Src	
Description	The first sequence of bytes to compare.
Type	Inline bytes (max 8192 bytes) or `Buffer` or `LastRxPacket`.
Default	No default value; this parameter is mandatory.
Example	**`[ 0x00, 0x09, 0x00, 0xE0, 0x00 ]`** to compare the specified bytes with the bytes specified in Dst. **`Buffer`** to compare bytes in the user buffer with the bytes specified in Dst.

SrcOffset	
Description	Offset in the source data of the first byte to compare.
Range	0 to 8191.
Default	`0`
Example	**`0`** will compare from the beginning of the PHY header when `LastRxPacket` is specified or from offset 0 of the user buffer when `Buffer` is used. **`5`** will compare from the beginning of the MAC header when `LastRxPacket` is specified or from offset 5 of the user buffer when `Buffer` is used. **`15`** will compare from the beginning of the payload when `LastRxPacket` is specified or from offset 15 of the user buffer when `Buffer` is used.

Dst	
Description	The second sequence of bytes to compare.
Type	`Buffer` or `LastRxPacket`.
Default	No default value; this parameter is mandatory.
Example	**`Buffer`** to compare bytes defined in `Src` with data in the user buffer. **`LastRxPacket`** to compare bytes defined in `Src` with data contained in the last received packet.

DstOffset	
Description	Offset in the destination buffer of the first data byte to compare.
Range	0 to 2047.
Default	No default value; this parameter is mandatory.
Example	**`0`** will compare from offset 0 of the user buffer when `Buffer` or `LastRxPacket` is used. **`200`** will compare from offset 200 of the user buffer when `Buffer` or `LastRxPacket` is used.

Mask	
Description	Mask to apply on each byte of the data to compare. The mask is applied with an AND operator.
Type	Inline bytes (max 8192 bytes).
Default	`0xFF` for all bytes specified in Data.
Example	**`[ 0x0F, 0x0F, 0xFF, 0xF0, 0xFF ]`** will use these bytes for the mask.

Length	
Description	Length of the data to compare.
Range	0 to 2047.
Default	No default value; this parameter is mandatory.
Example	**`5`** will copy 5 bytes.

5.9 WaitButtonPressed Instruction

The `WaitButtonPressed` instruction waits on user action on the specified button of the trigger board.

Example

```
WaitButtonPressed(
        Index                  => 0,
        Timeout                => 10s);
```

Parameter List

Index	
Description	Selects the button to wait on.
Range	0 to 1.
Default	No default value; this parameter is mandatory.
Example	`0` will wait until Button0 is pressed on the trigger board. `1` will wait until Button1 is pressed on the trigger board.

Timeout	
Description	Timeout after which the instruction is aborted.
Type	Time expressed in 60 MHz clock cycles or seconds.
Range	0 to 4,294,967,295 clock cycles or 0 to 71 seconds with a precision of 16.$\overline{6}$6 nanoseconds.
Default	No default value; this parameter is mandatory.
Example	`1.32ms` means 1,320 microseconds or 79,200 clock cycles. `620ns` will be floored down to 37 clock cycles. `3960clk` means 3,960 clock cycles or 66 microseconds. `1000` (without unity) is not allowed and will generate a warning.

5.10 WaitTriggerIn Instruction

The `WaitTriggerIn` instruction waits on the specified input of the trigger board.

Example

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

Parameter List

Input	
Description	Selects the input on which the condition should be waited on.
Range	`Any`, `BncIn`, `DigitalIn0` to `DigitalIn3`.
Default	No default value; this parameter is mandatory.
Example	**`Any`** waits on any inputs of the trigger board. **`BncIn`** waits on the BNC input of the trigger board.

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 condition.

Timeout	
Description	Timeout after which the instruction is aborted.
Type	Time expressed in 60 MHz clock cycles or seconds.
Range	0 to 4,294,967,295 clock cycles or 0 to 71 seconds with a precision of 16.6$\overline{6}$ nanoseconds.
Default	No default value; this parameter is mandatory.
Example	`1.32ms` means 1,320 microseconds or 79,200 clock cycles. `620ns` will be floored down to 37 clock cycles. `3960clk` means 3,960 clock cycles or 66 microseconds. `1000` (without unity) is not allowed and will generate a warning.

5.11 GenerateTriggerOut Instruction

The `GenerateTriggerOut` instruction generates a condition on the specified output of the trigger board.

Example

```
GenerateTriggerOut(
      Output              => BncOut,
      Mode                => PulseHigh);
```

Parameter List

Output	
Description	Selects the output to generate the trigger on.
Range	`All`, `BncOut`, `DigitalOut0` to `DigitalOut3`.
Default	No default value; this parameter is mandatory.
Example	**`All`** generates the condition on all outputs of the trigger board. **`BncOut`** generates the condition on the BNC output.

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 on the output. **`Toggle`** inverts the current level of the output.

5.12 ConfigureGenerator Instruction

The `ConfigureGenerator` instruction configures the generator in host or device mode.

Example

```
ConfigureGenerator(
        Mode             => Device,
        Speed            => HighSpeed);

ConfigureGenerator(
        Mode             => Host,
        Speed            => Chirp);
```

Parameter List

Mode	
Description	Specifies the generator mode.
Range	`Host`, `Device` or `ErrorInjection`.
Default	No default value; this parameter is mandatory.
Example	**`Host`** will configure the hardware for Host emulation. **`Device`** will configure the hardware for Device emulation. **`ErrorInjection`** will configure the hardware for error injection.

Speed	
Description	Specifies the link speed.
Range	`LowSpeed`, `FullSpeed`, `HighSpeed` or `Chirp`.
Default	No default value; this parameter is mandatory.
Example	**`LowSpeed`** will configure the link for low speed (1.5 Mbit/s). **`FullSpeed`** will configure the link for full speed (12 Mbit/s). **`HighSpeed`** will configure the link for high speed (480 Mbit/s). **`Chirp`** will configure the link for high speed chirp.

Results are undefined if `Mode` specifies `ErrorInjection` and `Speed` specifies `Chirp`.

5.13 ForceLinesState Instruction

The `ForceLinesState` instruction sets the link lines to the specified state.

Example

```
ForceLinesState(J);
ForceLinesState(State => SE0);
```

Parameter List

State	
Description	Specifies the state to be set on the link.
Range	`SE0`, `J` or `K`.
Default	No default value; this parameter is mandatory.
Example	**`SE0`** will wait until a SE0 state appears on the link. **`J`** will wait until a J state appears on the link. **`K`** will wait until a J state appears on the link.

5.14 ReleaseLinesState Instruction

The `ReleaseLinesState` instruction releases the link lines state.

Example

```
ReleaseLinesState();
```

Parameter List

This instruction does not take any parameters.

5.15 WaitLinesState Instruction

The `WaitLinesState` instruction waits for the specified lines state.

Example

```
WaitLinesState(
        State               => J,
        Timeout             => 20us);
```

Parameter List

State	
Description	Specifies the state to wait for.
Range	`SE0`, `J` or `K`.
Default	No default value; this parameter is mandatory.
Example	**`SE0`** will wait until a SE0 state appears on the link. **`J`** will wait until a J state appears on the link. **`K`** will wait until a J state appears on the link.

Timeout	
Description	Timeout after which the instruction is aborted.
Type	Time expressed in 60 MHz clock cycles or seconds.
Range	0 to 4,294,967,295 clock cycles or 0 to 71 seconds with a precision of $16.\overline{6}6$ nanoseconds.
Default	Waits for ever if not specified.
Example	**`1.32ms`** means 1,320 microseconds or 79,200 clock cycles. **`620ns`** will be floored down to 37 clock cycles. **`3960clk`** means 3,960 clock cycles or 66 microseconds. **`1000`** (without unity) is not allowed and will generate a warning.

5.16 SendPacket Instruction

The `SendPacket` instruction sends a raw USB packet using the mode and speed specified with the `ConfigureGenerator` instruction.

Example

```
SendPacket(
       RawData         => [ 0x69, 0x81, 0x58 ],
       Interval        => 17.554us,
       ComputeFcs      => false);

SendPacket(
       RawData         => Buffer,
       RawDataOffset   => 20,
       RawDataLength   => 12,
       Spacing         => 1us,
       ComputeFcs      => true);
```

Parameter List

RawData	
Description	Raw data of the packet to send including PID, payload and CRC.
Type	Inline bytes (min 1 byte, max 8191 bytes) or `Buffer`.
Default	No default value; this parameter is mandatory.
Example	**`[ 0x69, 0x81, 0x58 ]`** to use these bytes for the instruction. **`Buffer`** to use bytes from the user buffer.

RawDataLength	
Description	Length of the Buffer.
Range	1 to 8191.
Default	No default value; this parameter is mandatory when `Buffer` is used in `RawData`. This parameter cannot be used when inline bytes are specified in `RawData`.
Example	`1` will sent a packet with a length of one byte. `500` will sent a packet with a length of 500 bytes.

RawDataOffset	
Description	Offset of the data bytes in the Buffer.
Range	0 to 8191.
Default	No default value; this parameter is mandatory when `Buffer` is used in `RawData`. This parameter cannot be used when inline bytes are specified in `RawData`.
Example	**`0`** will send a packet from offset 0 in the Buffer. **`1024`** will send a packet from offset 1024 in the Buffer.

PrefixPid	
Description	PID to be added to the specified data.
Range	0 to 255.
Default	No default value; the packet will not be prefixed with a PID if this parameter is not specified.
Example	**`0xC3`** will send a `DATA0` PID.

Interval	
Description	Delay between the beginning of this instruction and the beginning of the next instruction.
Type	Time expressed in 60 MHz clock cycles or seconds.
Range	0 to 4,294,967,295 clock cycles or 0 to 71 seconds with a precision of $16.6\overline{6}$ nanoseconds.
Default	`0`.
Example	**`1.32ms`** means 1,320 microseconds or 79,200 clock cycles. **`620ns`** will be floored down to 37 clock cycles. **`3960clk`** means 3,960 clock cycles or 66 microseconds. **`1000`** (without unity) is not allowed and will generate a warning.

Spacing	
Description	Delay between the end of this instruction and the beginning of the next instruction.
Type	Time expressed in 60 MHz clock cycles or seconds.
Range	0 to 4,294,967,295 clock cycles or 0 to 71 seconds with a precision of 16.6$\overline{6}$ nanoseconds.
Default	`0.`
Example	**`1.32ms`** means 1,320 microseconds or 79,200 clock cycles. **`620ns`** will be floored down to 37 clock cycles. **`3960clk`** means 3,960 clock cycles or 66 microseconds. **`1000`** (without unity) is not allowed and will generate a warning.

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	`False`
Example	**`True`** to replace the specified CRC bytes with the computed CRC. **`False`** to leave the specified CRC bytes as is.

5.17 WaitPacket Instruction

The `WaitPacket` instruction waits for a packet matching the specified criteria.

Example

```
WaitPacket(
        Timeout             => 1ms,
        MatchOnlyValidCrc => true);
```

Parameter List

MatchOnlyValidCrc	
Description	Specifies if the instruction will only match packets with a valid CRC.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will break the script if a match occurs and the FCS of the received packet is valid. **`False`** will break the script if a match occurs independently of the FCS value of the received packet.

StoreRxPacket	
Description	Specifies if the instruction will store the received packet into the `LastRxPacket` special register.
Type	Boolean (`True` or `False`).
Default	`True`
Example	**`True`** will store the received packet into the `LastRxPacket` special register. **`False`** will not store the received packet.

Timeout	
Description	Timeout after which the instruction is aborted.
Type	Time expressed in 60 MHz clock cycles or seconds.
Range	0 to 4,294,967,295 clock cycles or 0 to 71 seconds with a precision of $16.6\overline{6}$ nanoseconds.
Default	Waits for ever if not specified.
Example	`1.32ms` means 1,320 microseconds or 79,200 clock cycles. `620ns` will be floored down to 37 clock cycles. `3960clk` means 3,960 clock cycles or 66 microseconds. `1000` (without unity) is not allowed and will generate a warning.

5.18 WaitTokenPacket Instruction

The `WaitTokenPacket` instruction waits for a token packet matching the specified criteria.

Example

```
WaitTokenPacket(
       SetupPid          => True,
       DeviceAddress     => DevAddr,
       EndpointNumber    => 0,
       Timeout           => 100ms,
       MatchOnlyValidCrc => true);
```

Parameter List

OutPid	
Description	Specifies if a OUT token packet will match.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will match OUT tokens. **`False`** will not match OUT tokens.

InPid	
Description	Specifies if a IN token packet will match.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will match IN tokens. **`False`** will not match IN tokens.

SetupPid	
Description	Specifies if a SETUP token packet will match.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will match SETUP tokens. **`False`** will not match SETUP tokens.

PingPid	
Description	Specifies if a PING token packet will match.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will match PING tokens. **`False`** will not match PING tokens.

ExtPid	
Description	Specifies if an EXT token packet will match.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will match EXT tokens. **`False`** will not match EXT tokens.

DeviceAddress	
Description	Specifies the device address to match.
Range	0 to 127.
Default	Match all device addresses.
Example	**`4`** will match the specified tokens only if they are sent to device address 4. **`0`** will match the specified tokens only if they are sent to the default device address.

EndpointNumber	
Description	Specifies the endpoint number to match.
Range	0 to 15.
Default	Match all endpoint numbers.
Example	`2` will match the specified tokens only if they are sent to endpoint number 2. `0` will match the specified tokens only if they are sent to the default control endpoint number.

MatchOnlyValidCrc	
Description	Specifies if the instruction will only match packets with a valid CRC.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will match only if the FCS of the received packet is valid. **`False`** will match all packets independently of their FCS value.

Timeout	
Description	Timeout after which the instruction is aborted.
Type	Time expressed in 60 MHz clock cycles or seconds.
Range	0 to 4,294,967,295 clock cycles or 0 to 71 seconds with a precision of $16.\overline{6}6$ nanoseconds.
Default	Waits for ever if not specified.
Example	**`1.32ms`** means 1,320 microseconds or 79,200 clock cycles. **`620ns`** will be floored down to 37 clock cycles. **`3960clk`** means 3,960 clock cycles or 66 microseconds. **`1000`** (without unity) is not allowed and will generate a warning.

5.19 WaitDataPacket Instruction

The `WaitDataPacket` instruction waits for a data packet matching the specified criteria.

Example

```
WaitDataPacket(
       Data0             => CurrentDataToggle,
       Data1             => !CurrentDataToggle,
       Timeout           => 1ms,
       MatchOnlyValidCrc => true);
```

ParameterList

Data0Pid	
Description	Specifies if a DATA0 packet will match.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will match DATA0 packets. **`False`** will not match DATA0 packets.

Data1Pid	
Description	Specifies if a DATA1 packet will match.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will match DATA1 packets. **`False`** will not match DATA1 packets.

Data2Pid	
Description	Specifies if a DATA2 packet will match.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will match DATA2 packets. **`False`** will not match DATA2 packets.

MDataPid	
Description	Specifies if a MDATA packet will match.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will match MDATA packets. **`False`** will not match MDATA packets.

MatchOnlyValidCrc	
Description	Specifies if the instruction will only match packets with a valid CRC.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will match only if the FCS of the received packet is valid. **`False`** will match all packets independently of their FCS value.

Timeout	
Description	Timeout after which the instruction is aborted.
Type	Time expressed in 60 MHz clock cycles or seconds.
Range	0 to 4,294,967,295 clock cycles or 0 to 71 seconds with a precision of $16.6\overline{6}$ nanoseconds.
Default	Waits for ever if not specified.
Example	**`1.32ms`** means 1,320 microseconds or 79,200 clock cycles. **`620ns`** will be floored down to 37 clock cycles. **`3960clk`** means 3,960 clock cycles or 66 microseconds. **`1000`** (without unity) is not allowed and will generate a warning.

5.20 WaitHandshakePacket Instruction

The `WaitHandshakePacket` instruction waits for a handshake packet matching the specified criteria.

Example

```
WaitHandshakePacket(
       AckPid            => True,
       NakPid            => True,
       Timeout           => 20ms,
       MatchOnlyValidCrc => true);
```

Parameter List

AckPid	
Description	Specifies if an ACK handshake packet will match.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will match ACK handshakes. **`False`** will not match ACK handshakes.

NakPid	
Description	Specifies if a NAK handshake packet will match.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will match NAK handshakes. **`False`** will not match NAK handshakes.

StallPid	
Description	Specifies if a STALL handshake packet will match.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will match STALL handshakes. **`False`** will not match STALL handshakes.

NyetPid	
Description	Specifies if a NYET handshake packet will match.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will match NYET handshakes. **`False`** will not match NYET handshakes.

ErrPid	
Description	Specifies if an ERR handshake packet will match.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will match ERR handshakes. **`False`** will not match ERR handshakes.

Timeout	
Description	Timeout after which the instruction is aborted.
Type	Time expressed in 60 MHz clock cycles or seconds.
Range	0 to 4,294,967,295 clock cycles or 0 to 71 seconds with a precision of $16.6\overline{6}$ nanoseconds.
Default	Waits for ever if not specified.
Example	**`1.32ms`** means 1,320 microseconds or 79,200 clock cycles. **`620ns`** will be floored down to 37 clock cycles. **`3960clk`** means 3,960 clock cycles or 66 microseconds. **`1000`** (without unity) is not allowed and will generate a warning.

5.21 WaitAndSendPacket Instruction

The `WaitAndSendPacket` instruction waits for a packet matching the specified criteria. If the expected packet is received the instruction will send the specified packet.

Example

```
SendPacket(
        RawData          => Buffer,
        RawDataOffset    => TokenPacketBufferOffset,
        RawDataLength    => TokenPacketSize,
        ComputeCrc       => True);

WaitAndSendPacket(
        RxTimeout              => TransactionTimeout,
        RxMatchOnlyValidCrc => True,
        TxRawData              => [ pidACK ],
        SendIfData0Pid         => True,
        SendIfData1Pid         => True);
```

Parameter List

RxDeviceAddress	
Description	Specifies the device address to match.
Range	0 to 127.
Default	Match all device addresses.
Example	`4` will match the specified tokens only if they are sent to device address 4. `0` will match the specified tokens only if they are sent to the default device address.

RxEndpointNumber	
Description	Specifies the endpoint number to match.
Range	0 to 15.
Default	Match all endpoint numbers.
Example	`2` will match the specified tokens only if they are sent to endpoint number 2. `0` will match the specified tokens only if they are sent to the default control endpoint number.

RxMatchOnlyValidCrc	
Description	Specifies if the instruction will only match packets with a valid CRC.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will match only if the FCS of the received packet is valid. **`False`** will match all packets independently of their FCS value.

RxTimeout	
Description	Timeout after which the instruction is aborted.
Type	Time expressed in 60 MHz clock cycles or seconds.
Range	0 to 4,294,967,295 clock cycles or 0 to 71 seconds with a precision of $16.6\overline{6}$ nanoseconds.
Default	Waits for ever if not specified.
Example	**`1.32ms`** means 1,320 microseconds or 79,200 clock cycles. **`620ns`** will be floored down to 37 clock cycles. **`3960clk`** means 3,960 clock cycles or 66 microseconds. **`1000`** (without unity) is not allowed and will generate a warning.

TxRawData	
Description	Raw data of the packet to send including PID, payload and CRC.
Type	Inline bytes (min 1 byte, max 8191 bytes) or `Buffer`.
Default	No default value; this parameter is mandatory.
Example	**`[ 0x69, 0x81, 0x58 ]`** to use these bytes for the instruction. **`Buffer`** to use bytes from the user buffer.

TxRawDataLength	
Description	Length of the Buffer.
Range	1 to 8191.
Default	No default value; this parameter is mandatory when `Buffer` is used in `RawData`. This parameter cannot be used when inline bytes are specified in `RawData`.
Example	**`1`** will sent a packet with a length of one byte. **`500`** will sent a packet with a length of 500 bytes.

TxRawDataOffset	
Description	Offset of the data bytes in the Buffer.
Range	0 to 8191.
Default	No default value; this parameter is mandatory when `Buffer` is used in `RawData`. This parameter cannot be used when inline bytes are specified in `RawData`.
Example	**`0`** will send a packet from offset 0 in the Buffer. **`1024`** will send a packet from offset 1024 in the Buffer.

TxPrefixPid	
Description	PID to be added to the specified data.
Range	0 to 255.
Default	No default value; the packet will not be prefixed with a PID if this parameter is not specified.
Example	**`0xC3`** will send a `DATA0` PID.

TxInterval	
Description	Delay between the beginning of this instruction and the beginning of the next instruction.
Type	Time expressed in 60 MHz clock cycles or seconds.
Range	0 to 4,294,967,295 clock cycles or 0 to 71 seconds with a precision of $16.6\overline{6}$ nanoseconds.
Default	`0`.
Example	**`1.32ms`** means 1,320 microseconds or 79,200 clock cycles. **`620ns`** will be floored down to 37 clock cycles. **`3960clk`** means 3,960 clock cycles or 66 microseconds. **`1000`** (without unity) is not allowed and will generate a warning.

TxSpacing	
Description	Delay between the end of this instruction and the beginning of the next instruction.
Type	Time expressed in 60 MHz clock cycles or seconds.
Range	0 to 4,294,967,295 clock cycles or 0 to 71 seconds with a precision of $16.6\overline{6}$ nanoseconds.
Default	`0`.
Example	**`1.32ms`** means 1,320 microseconds or 79,200 clock cycles. **`620ns`** will be floored down to 37 clock cycles. **`3960clk`** means 3,960 clock cycles or 66 microseconds. **`1000`** (without unity) is not allowed and will generate a warning.

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

WaitOutPid	
Description	Specifies if a OUT token packet will match.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will match OUT tokens. **`False`** will not match OUT tokens.

WaitInPid	
Description	Specifies if a IN token packet will match.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will match IN tokens. **`False`** will not match IN tokens.

WaitSetupPid	
Description	Specifies if a SETUP token packet will match.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will match SETUP tokens. **`False`** will not match SETUP tokens.

WaitPingPid	
Description	Specifies if a PING token packet will match.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will match PING tokens. **`False`** will not match PING tokens.

WaitExtPid	
Description	Specifies if an EXT token packet will match.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will match EXT tokens. **`False`** will not match EXT tokens.

WaitData0Pid	
Description	Specifies if a DATA0 packet will match.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will match DATA0 packets. **`False`** will not match DATA0 packets.

WaitData1Pid	
Description	Specifies if a DATA1 packet will match.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will match DATA1 packets. **`False`** will not match DATA1 packets.

WaitData2Pid	
Description	Specifies if a DATA2 packet will match.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will match DATA2 packets. **`False`** will not match DATA2 packets.

WaitMDataPid	
Description	Specifies if a MDATA packet will match.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will match MDATA packets. **`False`** will not match MDATA packets.

WaitAckPid	
Description	Specifies if an ACK handshake packet will match.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will match ACK handshakes. **`False`** will not match ACK handshakes.

WaitNakPid	
Description	Specifies if a NAK handshake packet will match.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will match NAK handshakes. **`False`** will not match NAK handshakes.

WaitStallPid	
Description	Specifies if a STALL handshake packet will match.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will match STALL handshakes. **`False`** will not match STALL handshakes.

WaitNyetPid	
Description	Specifies if a NYET handshake packet will match.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will match NYET handshakes. **`False`** will not match NYET handshakes.

WaitErrPid	
Description	Specifies if an ERR handshake packet will match.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will match ERR handshakes. **`False`** will not match ERR handshakes.

SendIfOutPid	
Description	Specifies if the packet will be sent when a OUT PID is received.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will send the packet when a OUT PID is received. **`False`** will not send the packet when a OUT PID is received.

SendIfInPid	
Description	Specifies if the packet will be sent when a IN PID is received.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will send the packet when a IN PID is received. **`False`** will not send the packet when a IN PID is received.

SendIfSetupPid	
Description	Specifies if the packet will be sent when a SETUP PID is received.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will send the packet when a SETUP PID is received. **`False`** will not send the packet when a SETUP PID is received.

SendIfPingPid	
Description	Specifies if the packet will be sent when a PING PID is received.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will send the packet when a PING PID is received. **`False`** will not send the packet when a PING PID is received.

SendIfExtPid	
Description	Specifies if the packet will be sent when a EXT PID is received.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will send the packet when a EXT PID is received. **`False`** will not send the packet when a EXT PID is received.

SendIfData0Pid	
Description	Specifies if the packet will be sent when a DATA0 PID is received.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will send the packet when a DATA0 PID is received. **`False`** will not send the packet when a DATA0 PID is received.

SendIfData1Pid	
Description	Specifies if the packet will be sent when a DATA1 PID is received.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will send the packet when a DATA1 PID is received. **`False`** will not send the packet when a DATA1 PID is received.

SendIfData2Pid	
Description	Specifies if the packet will be sent when a DATA2 PID is received.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will send the packet when a DATA2 PID is received. **`False`** will not send the packet when a DATA2 PID is received.

SendIfMDataPid	
Description	Specifies if the packet will be sent when a MDATA PID is received.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will send the packet when a MDATA PID is received. **`False`** will not send the packet when a MDATA PID is received.

SendIfAckPid	
Description	Specifies if the packet will be sent when an ACK PID is received.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will send the packet when an ACK PID is received. **`False`** will not send the packet when an ACK PID is received.

SendIfNakPid	
Description	Specifies if the packet will be sent when a NAK PID is received.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will send the packet when a NAK PID is received. **`False`** will not send the packet when a NAK PID is received.

SendIfStallPid	
Description	Specifies if the packet will be sent when a STALL PID is received.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will send the packet when a STALL PID is received. **`False`** will not send the packet when a STALL PID is received.

SendIfNyetPid	
Description	Specifies if the packet will be sent when a NYET PID is received.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will send the packet when a NYET PID is received. **`False`** will not send the packet when a NYET PID is received.

SendIfErrPid	
Description	Specifies if the packet will be sent when an ERR PID is received.
Type	Boolean (`True` or `False`).
Default	`False`
Example	**`True`** will send the packet when an ERR PID is received. **`False`** will not send the packet when an ERR PID is received.

5.22 HostAutoGenerateSof Instruction

The `HostAutoGenerateSof` instruction starts or stops automatic Start-of-Frame generation.

Example

```
HostAutoGenerateSof();
HostAutoGenerateSof(False);
```

Parameter List

Enable	
Description	Specifies if SOF are automatically generated.
Type	Boolean (`True` or `False`).
Default	`True`
Example	**`True`** will automatically generate SOFs accordingly to the link speed. **`False`** will stop generating SOFs.

5.23 HostWaitGeneratedSof Instruction

The `HostWaitGeneratedSof` instruction synchronizes on a Start-of-Frame automatically generated by the hardware.

Example

```
HostWaitGeneratedSof(
        Timeout             => 150us);
```

Parameter List

Description	Timeout after which the instruction is aborted.
Type	Time expressed in 60 MHz clock cycles or seconds.
Range	0 to 4,294,967,295 clock cycles or 0 to 71 seconds with a precision of $16.\overline{6}6$ nanoseconds.
Default	Waits for ever if not specified.
Example	**`1.32ms`** means 1,320 microseconds or 79,200 clock cycles. **`620ns`** will be floored down to 37 clock cycles. **`3960clk`** means 3,960 clock cycles or 66 microseconds. **`1000`** (without unity) is not allowed and will generate a warning.

5.24 HostSetMaxTransactionDuration Instruction

The `HostSetMaxTransactionDuration` instruction specifies the expected time needed to send a transaction. The host processor will automatically check if this time is large enough to fit the current frame. If not, the transaction will be delayed until the beginning of the next frame.

Example

```
HostSetMaxTransactionDuration(Duration => 60us);
```

Parameter List

Duration	
Description	Computer duration of the transaction to be sent.
Type	Time expressed in 60 MHz clock cycles or seconds.
Range	0 to 4,294,967,295 clock cycles or 0 to 71 seconds with a precision of $16.6\overline{6}$ nanoseconds.
Default	No default value; this parameter is mandatory.
Example	**`0.66ms`** means 0,660 microseconds or 39,600 clock cycles. **`620ns`** will be floored down to 37 clock cycles. **`3960clk`** means 3,960 clock cycles or 66 microseconds. **`1000`** (without unity) is not allowed and will generate a warning.

5.25 HostResetMaxTransactionDuration Instruction

The `HostResetMaxTransactionDuration` instruction resets the duration specified by `HostSetMaxTransactionDuration`. After this instruction is called, the processor will not delay a transaction if it is too close to a frame boundary.

Example

```
HostResetMaxTransactionDuration();
```

Parameter List

This instruction does not take any parameters.

Frequently Asked Questions

Q **The USB Explorer 260 transmits data using a USB 2.0 connection. Do I need a USB 2.0 host controller?**

A Although the USB Explorer 260 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 260, please ensure it is connected on a high speed USB 2.0 enabled host controller before contacting technical support.

Q **I have got one host controller and I'd like to add a second one. How can I achieve this?**

A Installing a USB extension card is the easiest way to add a host controller to your computer. Furthermore, nowadays most of these extension cards are USB 2.0-compatible, which will enable you to wholly gain from all your analyzer's power. Talk to your local dealer about getting a USB 2.0 host controller card.

Q **What is the maximum amount of data that I can generate with the USB Explorer 260 Generator?**

A The Generator uses its internal memory and hard disk to store data to be generated. The maximum quantity of data is therefore limited by the size of the internal memory.

Q **Is it possible to upgrade the firmware of the USB Explorer 260?**

A Yes, the firmware is automatically updated with each new software release. No user intervention is required; the latest version of the firmware will be downloaded when you run the most recent version of the software.

Q **What can I connect to the large connector on the back of the product?**

A The Auxiliary Equipment connector enables hardware extensions. Several options are currently available and others may be provided in the future. Please contact the Ellisys sales team for more information.

Q **I cannot run the software installation file, why?**

A The software installation file requires Microsoft Windows Installer 3.0 or higher, which is available for download from the Microsoft web site.

Q **I would like to connect the USB analyzer on a EHCI host controller but it seems that it doesn't work. What can I do?**

A We took note that several EHCI drivers supplied by manufacturers of USB 2.0 add-in cards could cause problem. We strongly recommend the use of the Microsoft EHCI driver. You can find more information on the installation of this driver at: http://www.usbman.com/USB%202%20News.htm.

Need more help?

Go to the Ellisys web site 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.

Glossary

This glossary lists terminology terms, abbreviations and acronyms that you may come across while reading this User Guide and working with Ellisys products.

ACK	Acknowledgment code - Usually sent at the end of successful transaction.
Addr	Address - A field used to identify a given device.
Analyzer	An instrument that capture traffic exchanged between devices.
API	Application Programming Interface - A set of functions used by a program to communicate with another.
Bandwidth	The transmission capacity of an electronic pathway such as a communication line, computer bus or computer channel.
BIN	Binary - A representation of values that uses two symbols, typically 0 and 1.
BER	Bit Error Rate - The number of bits in error divided by the total number of bits.
BNC	Bayonet-Neill-Concelman - A connector for coaxial cables.
Bookmark	A stored location for quick retrieval at a later date.
bps	Bits per second - The measurement of the speed of data transfer in communication systems.
Breakpoint	The location in a program used to temporarily halt the program for testing and debugging.
Code Snippet	A small piece of program code usually used to guide the user.
CSV	Comma-separated Values - A delimited data format that has fields separated by the comma character and records separated by new lines.
Dec	Decimal - A representation of values that uses ten symbols, typically 0 to 9.
DUT	Device Under Test - A device that is being analyzed or debugged.
EDX	Ellisys index file - A file format used to index information found in another file.

ESE	Ellisys settings file - A file format used to store user settings.
EUI-48	Unique identifier partly assigned by the IEEE RAC and partly defined by the manufacturer of an equipment to uniquely identify a networking device.
FCS	Frame Check Sequence - A number added to a stream of information that is used for error detection.
FIFO	First In First Out - A storage method that retrieves first the item stored for the longest time.
Gbps	Gigabits per second - 1,073,741,824 bits per second.
GByte	Gigabytes - 1,073,741,824 bytes.
Hex	Hexadecimal - A representation of values that uses sixteen symbols, typically 0 to 9 and A to F.
Handshake	The resulting status of a data exchange.
Host	A computer that acts as a source of information or signals.
IDE-type connector	A type of electric connector usually attached to a flat ribbon cable.
LED	Light Emitting Diode - Display and lighting technology commonly used on electronic equipment to indicate their status.
Kbps	Kilobits per second - 1,024 bits per second.
KByte	Kilobytes - 1,024 bytes.
Loop	A repetition within a program or script.
Mbps	Megabits per second - 1,048,576 bits per second.
MByte	Megabytes - 1,048,576 bytes.
NAK	Negative Acknowledgement - An answer to a request that can express anything but acceptance.
Packet	A block of data that is transmitted over a communication link.
Payload	The actual data in a packet minus all headers attached for transport and minus all descriptive metadata.
Protocol	The format and procedures that govern the transmitting and receiving of data.
RX	A communication abbreviation for receive.
Script	A set of instructions that is executed without user interaction.

Snippet	A small piece of program code that guides the user in how to write a specific instruction.
SOF	Start of Frame - A packet used for USB time synchronization.
TX	A communication abbreviation for transmit.
USB	Universal Serial Bus - An interface that connects between a computer and peripheral devices (such as a keyboard, game controllers, telephone, printer, etc.).
XML	Extensible Markup Language - A reasonably human-legible structured language aimed to facilitate the sharing of data across heterogeneous information systems.

Index

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