LabVIEW Universal Transcriptor Quicstart Guide

Lubomir Jagos

In The Restaurant at the End of the Universe

2019

Table of Contents

LabVIEW Universal Transcriptor – Quickstart Guide

This documents try to minimaze time and easier process of “installation” LabVIEW Universal Transcriptor and to show you first steps.

Everything is made using packages

Packages are in LabVIEW to maintain code libraries so transcriptor and it’s modules are using it. It’s splitted into basic transcriptor package with name “LabVIEW to Arduino Transcriptor”, this package contains all system functions (basically it’s wrapper over basic Arduino libraries and functions). It allows user to do basic operations like add, multiplication, serial line output and read, work with cluster, structures (while loop, for loop) and more. This is core package so without it there is no transcription, no way how to get C++ code from LabVIEW.

You can find this and many other packages at this location:
https://github.com/LubomirJagos/LabVIEW-for-Arduino-Libraries-Packages

Examples repository

Starting point for every project. Why to reinvent wheel when you can borrow already tested ideas. I tried to create some basic examples to confirm right functionality of my implemented modules and to show how to use VI to have valid program which is transcriptor able to transcript into C++ code. So I developed them and placed them here: https://github.com/LubomirJagos/LabVIEW-Universal-Transcriptor-Examples

Installation

Transcriptor is builded as VIPM package, so all you have to do is just download it and install from this link https://github.com/LubomirJagos/LabVIEW-for-Arduino-Libraries-Packages/blob/master/lubomirjagos_lib_labview_universal_transcriptor-0.1.0.49.vip

After installation it should appear in your pallete menu item “LabVIEW Universal Transcriptor”, into this subpallete are installed also all aditional packages for transcriptor for different modules (accelerometer, ESP8266, DAC, …), transcriptor core module subpallete has name “LabVIEW to Arduino Transcriptor” as you can see on image below.

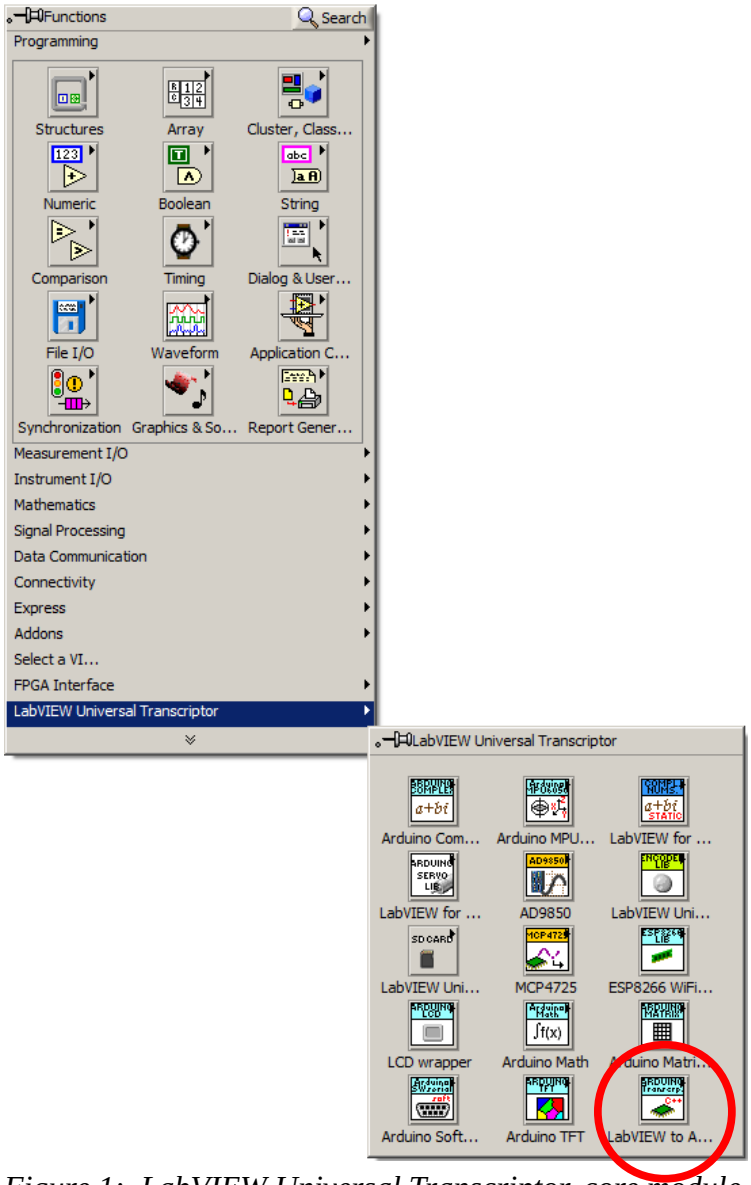

Figure 1: LabVIEW Universal Transcriptor, core module in pallete

This subpallete contains basic Arduino functions as write and read from digital and analog IO pins, setting interruptions, some timing function for delay and then some basic programming elements as loops, clusters, case structures, conditional expressions and so,as you can see on image below.

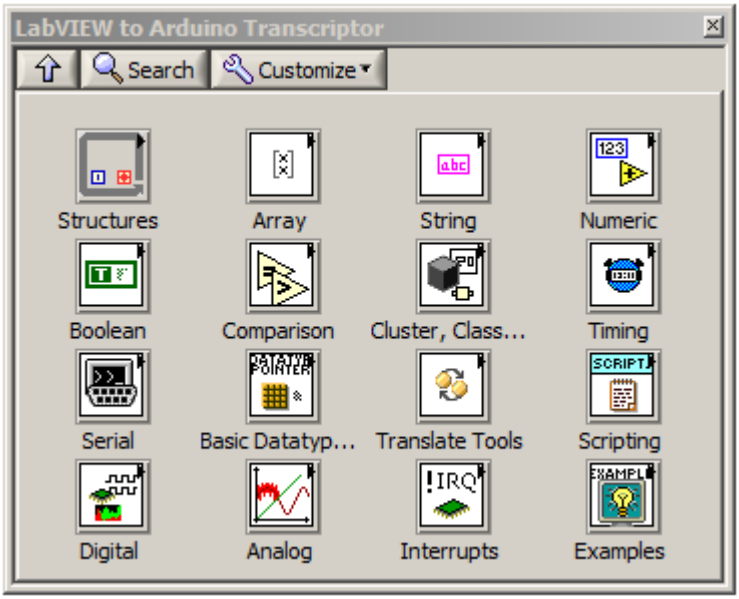

Figure 2: Basic Arduino and programming elements which are part of transcriptor core module

So this was about palletes and functions which is transcriptor able to translate from LabVIEW to C++ for Arduino. To make translation process automatic, transcriptor core package also install into tools it's application for transcripting VIs into C++ for Arduino, see image below.

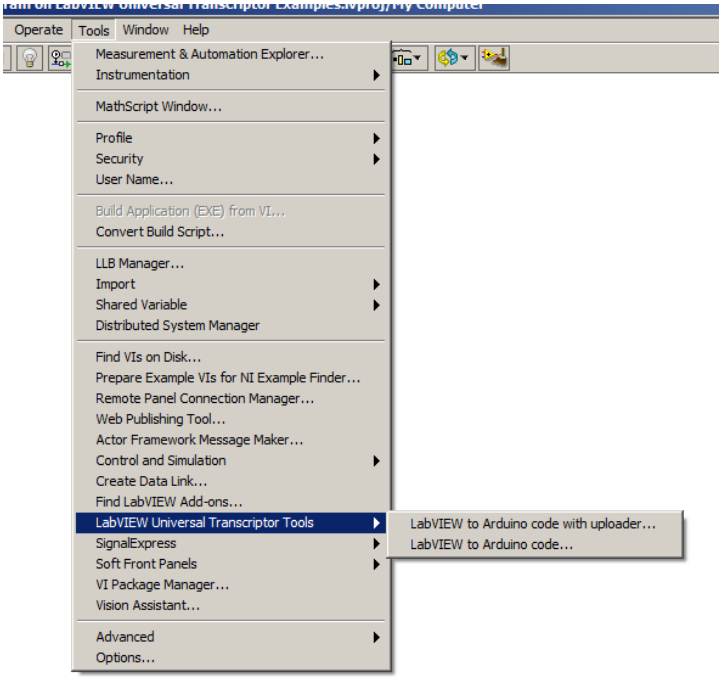

Figure 3: Transcriptor tools installed in LabVIEW IDE

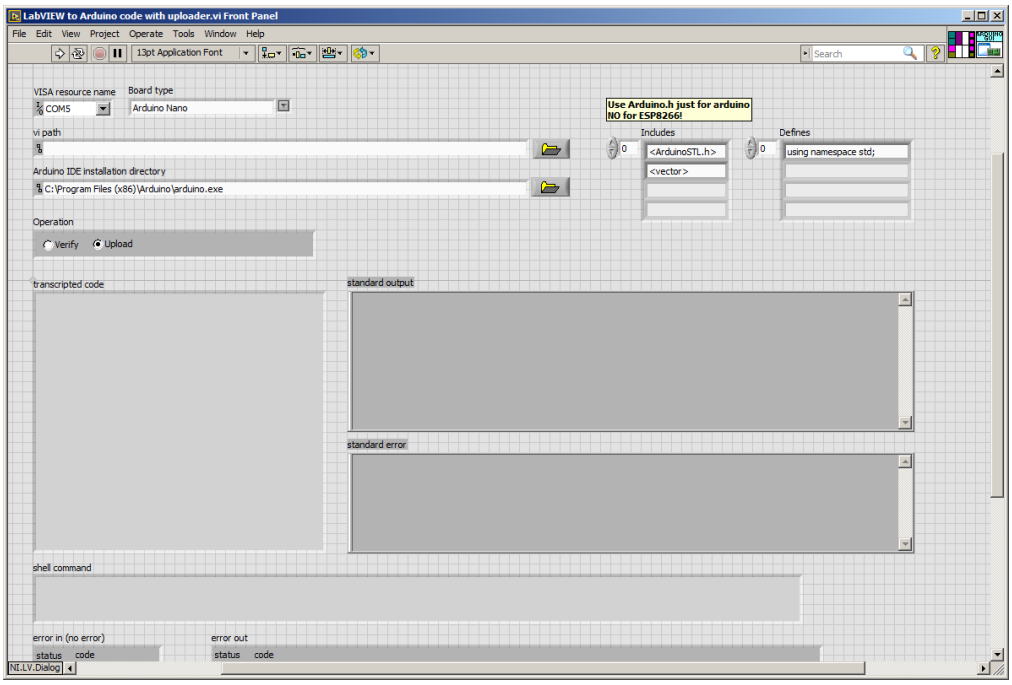

Figure 4: Point and click transcriptor tool to get your LabVIEW program into Arduino.

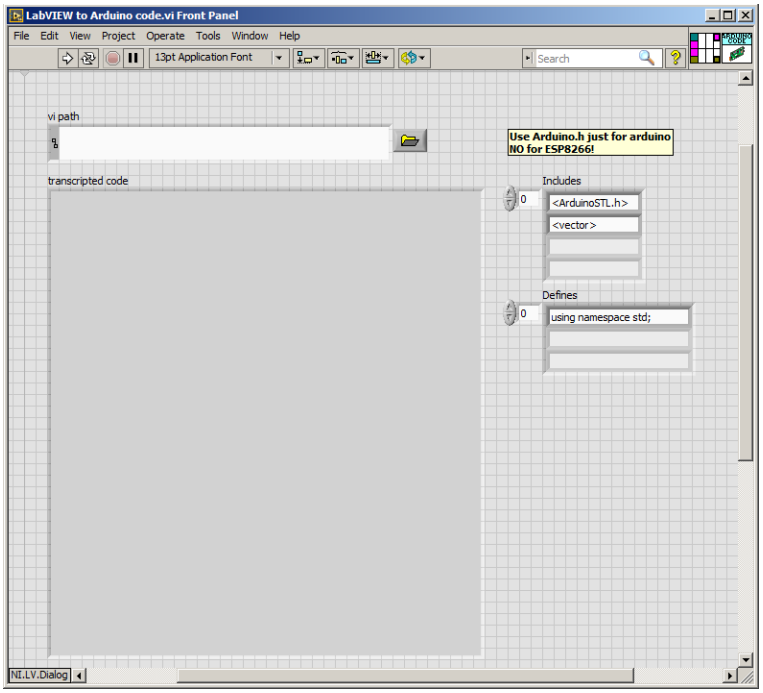

Figure 5: Basic transcriptor just to generate C++ code for Arduino without uploader.

Start using transcriptor

Using this transcriptor pretty straightforward. As beginner example let’s make some blinking LED . You can find at my github repository with examples and there is also Blink demo: https://github.com/LubomirJagos/LabVIEW-Universal-Transcriptor-Examples/tree/master/Blink

So clone this repository or download it and open example.

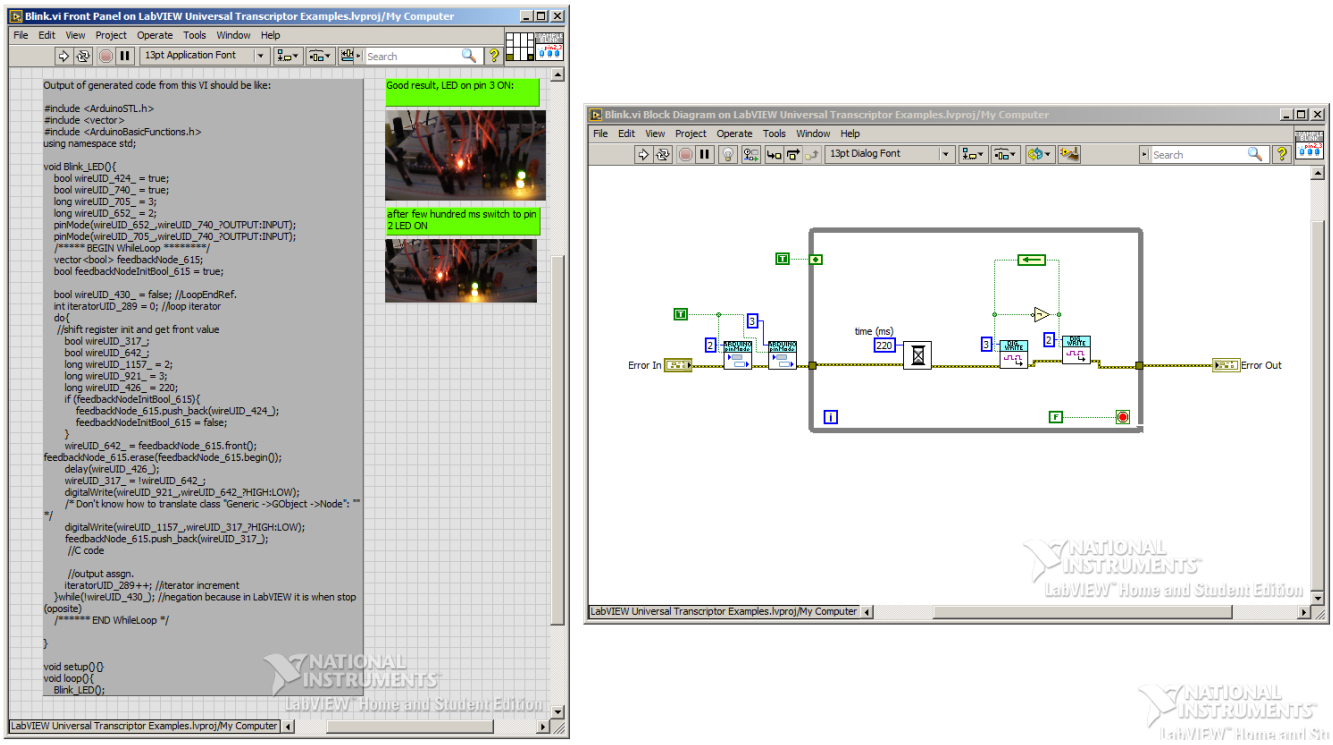

Figure 6: Blink example, after some hundreds milisecond arduino toggle digital pins 2 and 3, so LEDs are blinking.

Extending transcriptor

This transcriptor was written with idea to be etensible in mind, so there is mechanism how to interface any C++ function into it.

There are just simply rules which user has to follow to sucessuflly add new interface for C++ code into it, best way to understand hwo to do that is describe how transcriptor is translating code from LabVIEW diagram into code:

1. Transcriptor is traversing diagram using error wire as leading wire which is responsible for VI executing order
2. When hit VI it's looking into it's directory for VI named "…Translator… .vi"
3. If VI's local transcriptor VI is found it's invoked with input params (VI inputs names and datatypes, parent VI reference, ….)
4. Resulting C++ code string is taken and used further to generate complete code

So as you can see if you want to interface new Arduino C++ library to transcriptor you need to create folder with VIs:

1. VIs which represent Arduino C++ functions (their block diagram can be empty), thei HAVE TO BE password protected (this is what differentiate them from normal VIs)
2. transcriptor VI named "[anything] Translator [anything].vi", this VI contains actual strings which are place in code on VI place in diagrams

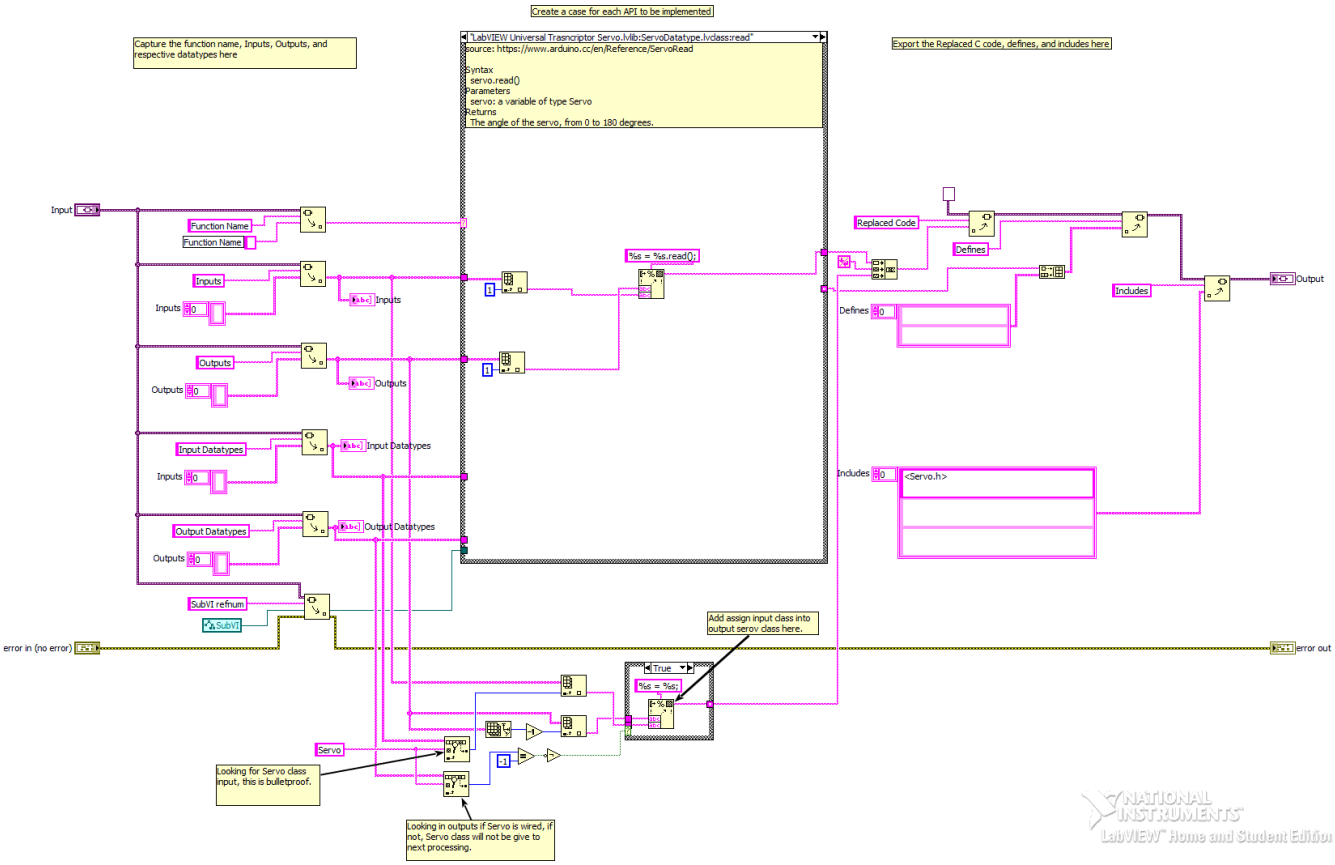

Figure 7: Example of translator VI from ServoMotor library, user has to define just part of diagram inside case structure, each case is for different user defined interface VI

Next steps

This was beginner manual to guide you through basics of this transcriptor, now you should be able to play with Arduino blink some LEDs, what is great beginning to make something bigger. All big things consists from smaller things and way to understand complex systems is to study them from both direction top and bottom, hopefully this transcriptor will be useful tool for you and will help you to programme your desired system for Arduino using LabVIEW.

I prepared more examples to cover sensor libraries which I developed for most used sensors for Arduino. Don’t hesitate and take look :)