# visual_programming Documentation

***Release 0.1***

**Jinesh Kallunkathariyil**

**Jul 16, 2019**

# Contents

CHAPTER 1

# Introduction

Let's examine the workings of the program with a simple example as shown in Gif 1.1. In the gif, we have five blocks, a Main block, Array block, Duplicate block, Expression block and Graph block. The number suffix is used to have a unique name for every block, in case of multiple blocks of the same type. The main block calls an array block which creates an array data. This array data is duplicated into two identical arrays by the duplicate block. One data is fed through an expression block which has a sin(x) expression in this example, produce the sine data. Finally both data are fed to the graph block and plotted. Thus this program generate a sine(x) data and plot it.

Fig. 1: Gif 1.1: An animated program and data flow, a simple example of visual programming. The Main block is the entry block, which calls an Array block. The Array block creates an array data and this data is sent to the Duplicate block. The Duplicate block duplicates the data and the output is two identical arrays. One is fed through an Expression block, in this case a sin(x) expression. Finally data are displayed in a Graph block.

# CHAPTER 2

# Block diagram description of generated program

Now that we know what the program does, let's better understand how the software do it with the help of Figure 1.1.

The entry point to the program is the main block. Main block generates two source codes (main.c and eventLoop.c) and one header file (eventLoop.h). The main.c has the entry main() function defined. From within the main() function, it calls exec() defined in the eventLoop.c and pass the name of the next block to be called. In this case, the name of the next block is Array_0. The exec() function has two jobs, the first is to create a data pack with the name of the next function block to be called (the first block connected from main block output) and send it to the linked list. The second job is to call the eventloop thread which loops continuously till the end of the program life.

The event loop continuously scan the linked list for any available data and if it is available, it calls the block (function) to be executed with the name and pass the packed data.

In our example, each visual block are array block, duplicate block, expression block and graph block.

## 2.1 Data packing format

## 2.2 Linked list structure

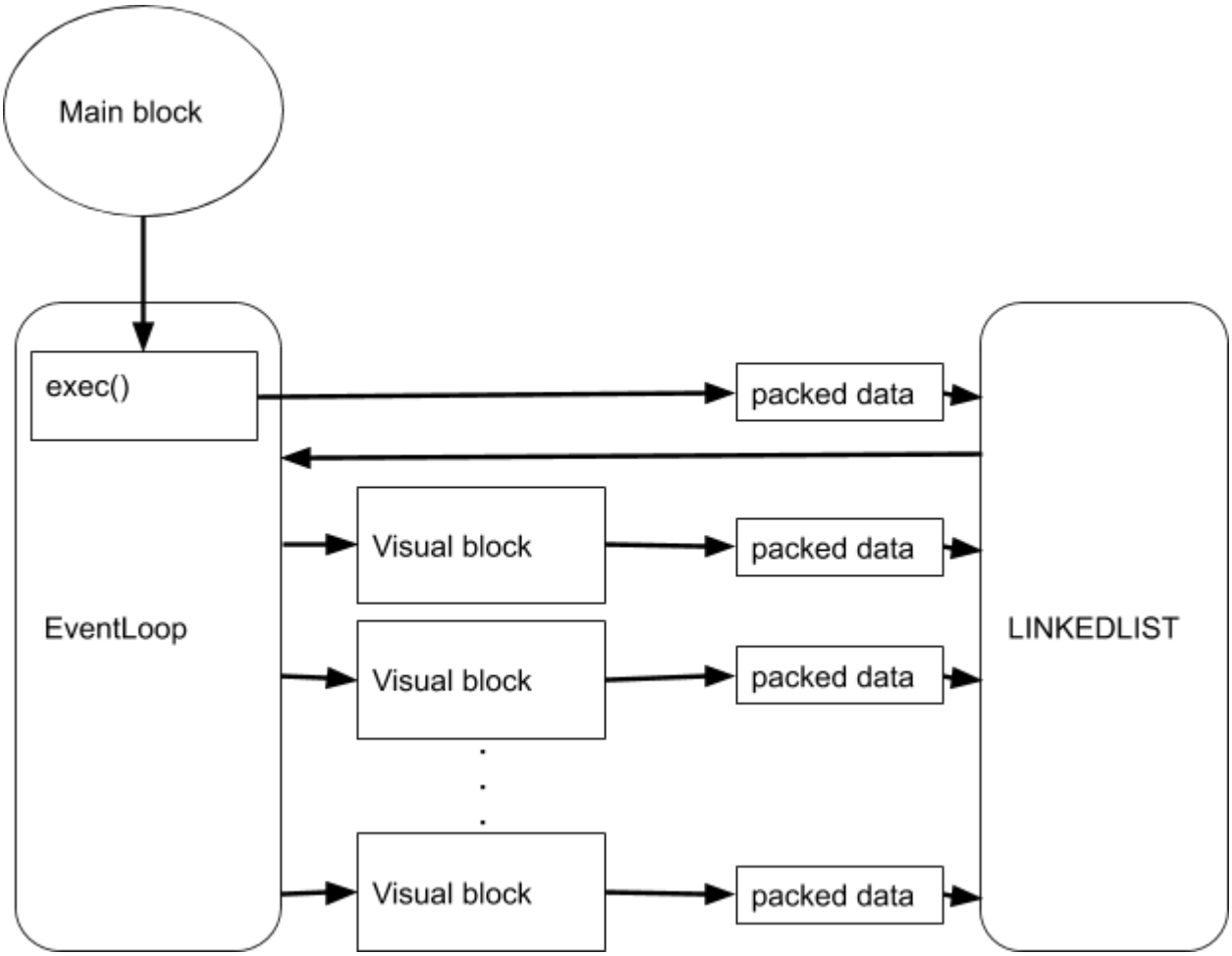


Fig. 1: Figure 1.1: A block diagram representation of the workings of the program. Each block is a C function. Main in the entry function which is defined in main.c. See text for more details.

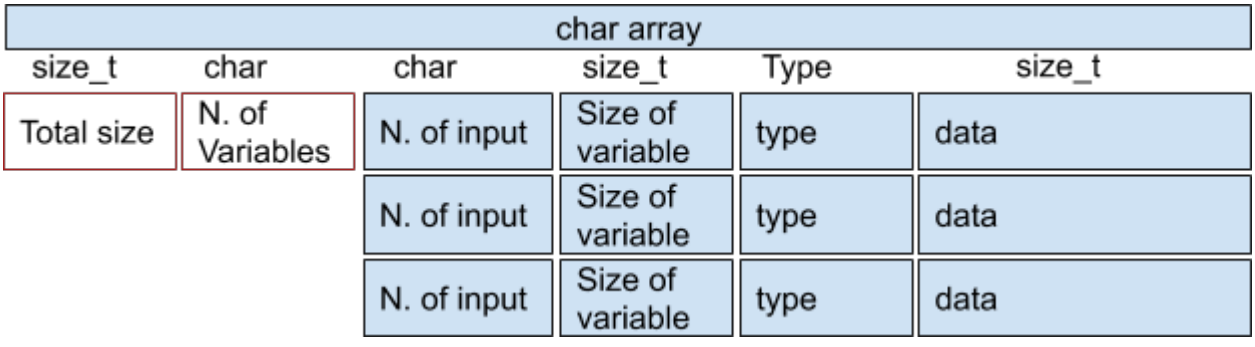


Fig. 2: Figure 1.2: Data packing format.

PData_t

```
typedef struct PData{
        char *funcName;
        size_t size;
        int nIns;
        int nInsFilled;
        char *in;
        struct PData *next;
} PData_t;
```

Fig. 3: Figure 1.3: Linked list structure.

# CHAPTER 3

# Target program generation and simulation

In the previous section, we saw an overview of how the generated program works. In this section, we will see the connection between visual programming environment, simulation and the generated program.

Initially, the user design the program by placing the blocks, connecting them and by editing the configuration and expressions. When he finishes the design and click the run button, few things happens. First a connection tree is made from the design, where the blocks are tree nodes with main block is the root of the tree. This connection tree has all the information about the connections between the blocks. Next, the code for each block is generated.

## 3.1 Code generation

In code generation face, the C code for each block is generated. These blocks are functions. The expressions on the blocks are tokenized and parsed and then compiled to the C code (desugared). More on this in later sections. A make file necessary for the compilation also generated.

## 3.2 Just in Time Compilation (JIT) of library

Just in time compilation is the compilation of program during the run time. In this case also we have the same situation. The codes generated from previous section is compiled into a library. THis is the same library which will be used in the visual programming environment, or on a stand alone application.

## 3.3 Dynamic loading of library

The next important feature used is the dynamic loading of library. The library compiled in the previous section is loaded by the environment and executed. It is called dynamic loading because the library is loaded at a time after the startup of the program.

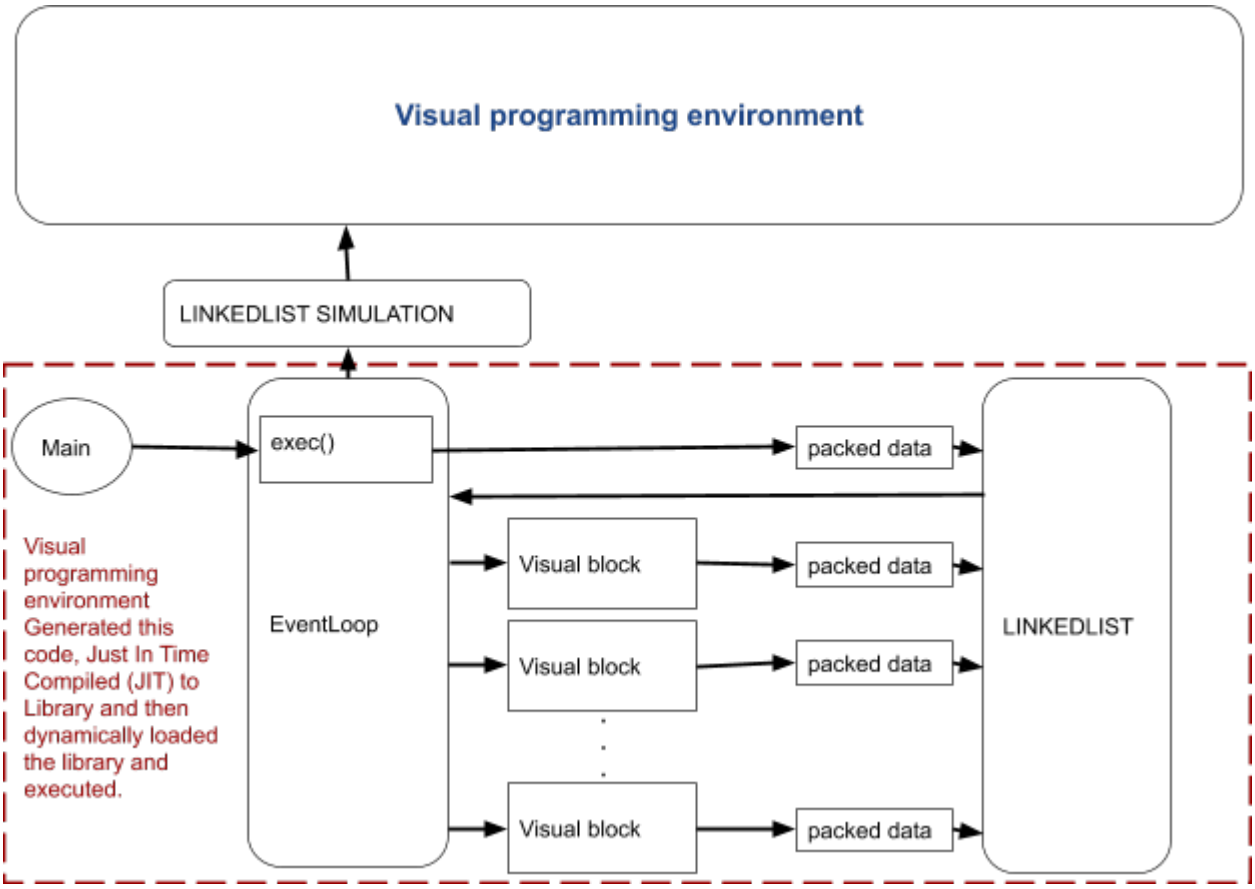


Fig. 1: Figure 1.4: Block diagram representation of connection between the visual programming environment and the generated library. The environment first desugar the expressions to C code and then generate the codes for each block. They are then compiled in just in time to a library. This library then dynamically loaded and executed. Each block data are copied into a second linked list for shown in the visual programming environment.

## 3.4 Data sharing from library to the visual environment

All blocks output data are copied into a second linked list (queue) by the event loop in the library, when the library called from the visual environment (in the case of stand alone execution of library), no data is copied into the second linked list. The visual environment has access to this linked list and it access the data in a FIFO manner. All these data are printed or plotted on corresponding block outputs in the visual environment. In this way, the visual debugging is simplified and user can easily verify each block output. Even the user can simulate the program flow in a step by step manner, since the data is still available in the linked list for simulation.

One important advantage is that the same library is used for the visual simulation and also for the stand alone application. This will ensure the same working.

## 3.5 Concurrent programming

Every block function are run in a thread, thus we can say it is a concurrent programming.

## 3.6 Distributed programming

The visual programming supports also distributed programming, which means run some blocks on one computer and other blocks on a different computer. The program communicates with ip address and port number. There are two blocks facilitates this functionality, network server block and network client block. They can be used to send data over network or for a remote procedure call (RPC).

CHAPTER 4

# Visual programming environment

In the previous sections, we were showing the visual programming environment as a single block, but there are many things going on in this block. In this section we will analyse the block in details.

## 4.1 Visual blocks

The idea of blocks for programming is coming from our nature of explaining or comprehending ideas by writing boards. This is how we explain the things to another person, our creating our ideas more clear. In this way, we will be concentrating on what to do rather than how to do. There are many blocks available, let's go through them now.

### 4.1.1 Main block

We already saw main block in previous sections. Main block is the entry block in the program as the main function in the C.

### 4.1.2 Array block

Array block is used to generate arrays. There are different ways to generate arrays. One is the classical programming way, where we give the total number of array elements, beginning (included), upper limit (excluded) and the type as shown in Figure 3.1.

Another way to generate the array is by the set notation (known as set comprehension or list comprehension). One can define it as, x <- [0,1 .. 99]

### 4.1.3 Duplicate block

This block duplicates the data into two. For example, if we have one data connection from the array block, we can duplicate the data and have two similar data outputs from duplicate block.

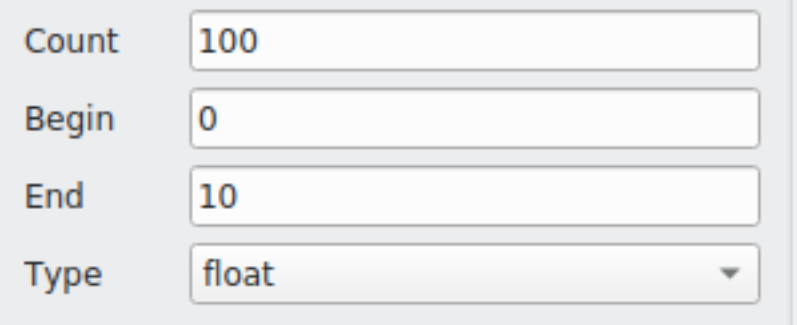


Fig. 1: Figure 3.1: Array definition.

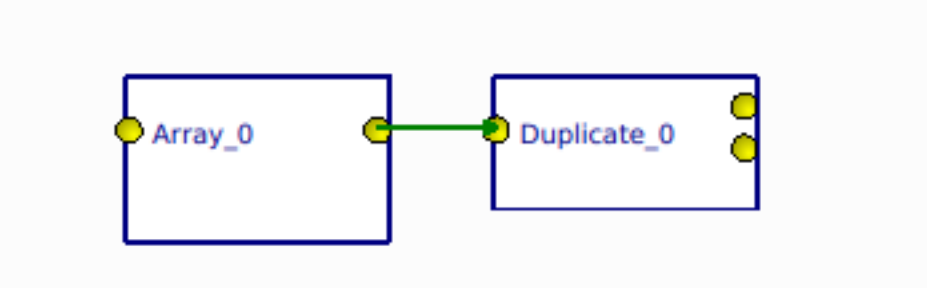


Fig. 2: Figure 3.2: Data duplicate block with array block.

### 4.1.4 Expression block

In expression block, one can define expressions which will act on the input data. For example if one wants the sin(x) function acting on the input, just write the sin(x) function in the expression input. The output will be a sin(x). One can also give some functions as an expression, for example the factorial function.

Graph block: A graph block plot the data coming to the input. In Figure 3.3, one can see the graph block connected to other blocks draw a sine function.

### 4.1.5 Network server block

A network server block is used for the distributed programming. One can call a remote procedure connected with the server block. Thus it can be also called RPC. The block receive data through a port and ip address from a Network client block. In Figure 3.4, a network server block connected to another blocks is shown.

### 4.1.6 Network client block

A network client block is used to send data over network and it can be considered as the RPC in distributed computing.

In Figure 3.6, shown both network server and client blocks are in action. The client block send data to the server block and the server block plot the data.

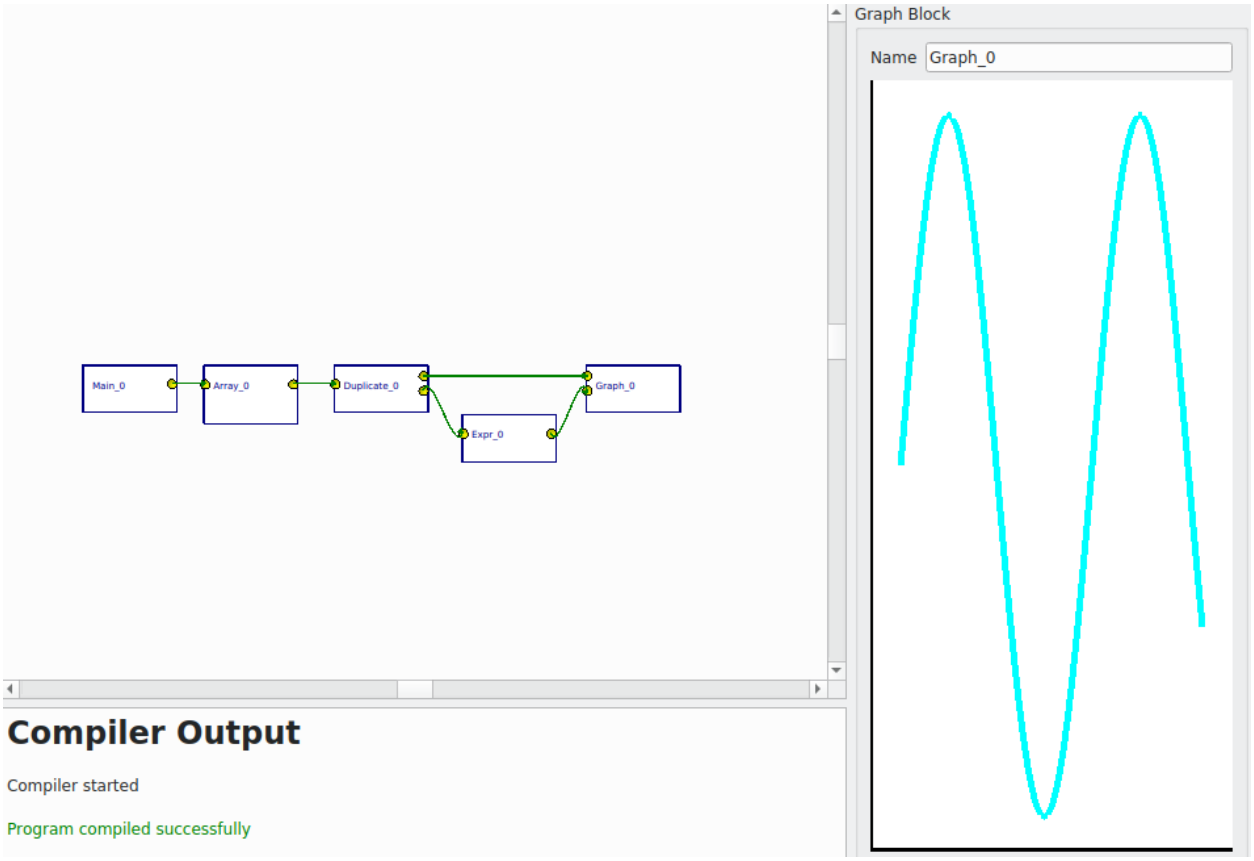


Fig. 3: Figure 3.3: A graph block connected to the other blocks plotted a sine function

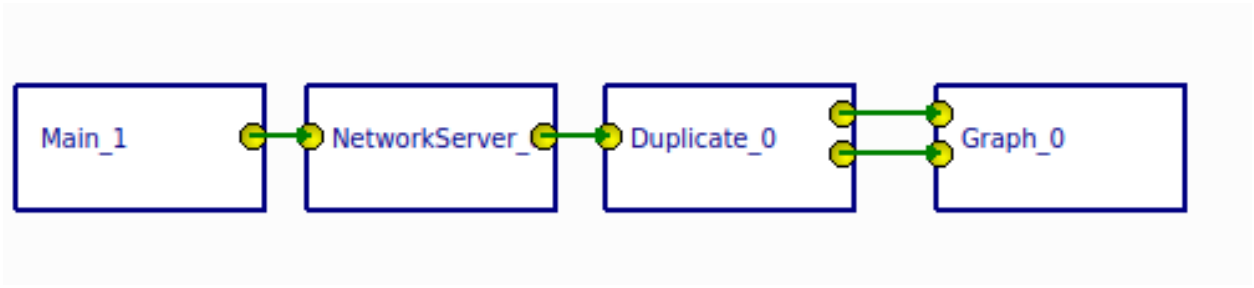


Fig. 4: Figure 3.4: A network server block in action. It receives data sent from network client block and plot it.

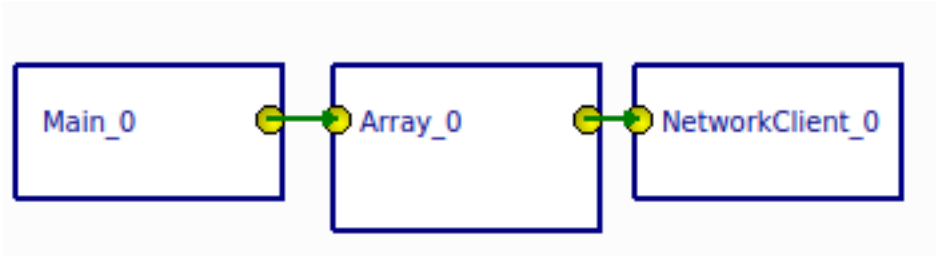


Fig. 5: Figure 3.5: A network client block send an array data over ip to specific port.

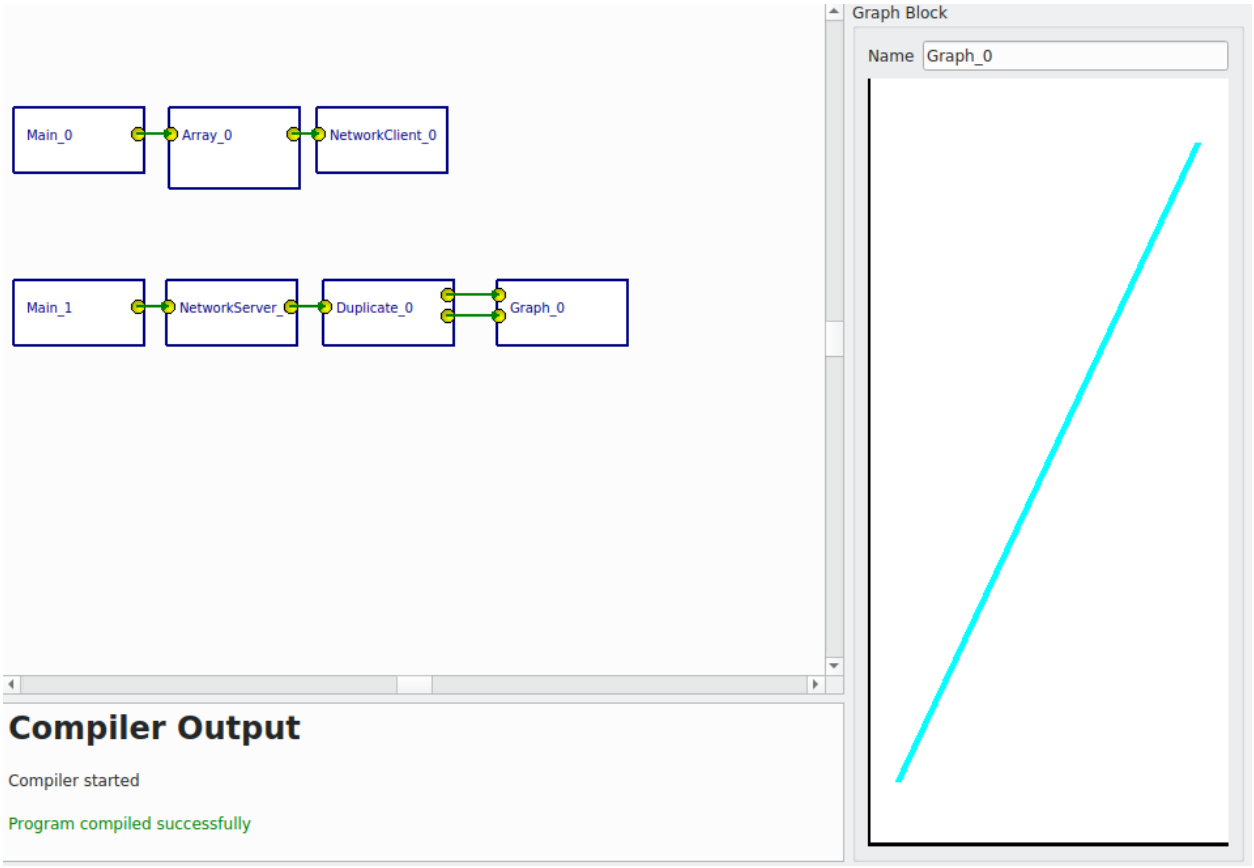


Fig. 6: Figure 3.6: A network client block and network server block in action.

### 4.1.7 State Machine block

### 4.1.8 Simulation1D block

The idea of simulation1D block is to simulated the nature of data in 1D. For example, if we give a sine wave input, sphere will oscillate. a simulation1D block is a block with one sphere and two inputs to it. One input is the time and another input is the position. The position can be assigned either in x - axis or y-axis or z-axis.

Fig. 7: Gif 3.1: An animated program and data flow to show the simulation block. The Main block is the entry block, which calls an Array block. The Array block creates an array data and this data is sent to the Duplicate block. The Duplicate block duplicates the data and the output is two identical arrays. One is fed through an Expression block, in this case a sin(x) expression. Finally data are simulated in a Simulation1D block.

### 4.1.9 GSLOdeSolver block

CHAPTER 5

# Expression parsing

The mathematical expressions written on the visual blocks configuration environment has to be compiled into the C language. For this purpose, first we need to tokenize the expression string with a lexical analyzer, then parse it with Earley recognizer and parser and then translate into C code. We will go through these steps in the following sections.

## 5.1 Lexical analyzer

A lexical analyzer is a program that separates the source code into a sequence of lexemes. It reads the input source code character by character, recognize the lexemes and outputs a sequence of tokens describing the lexemes. A good read on the lexical analyzer can be found on the link https://hackernoon.com/lexical-analysis-861b8bfe4cb0.

## 5.2 Regular expression

## 5.3 Context free grammar

## 5.4 Earley recognizer and parsing

A good read on the Earley recognizer can be found on Wikipedia link https://en.wikipedia.org/wiki/Earley_parser

## 5.5 Set comprehension (list comprehension)

CHAPTER 6

# Visual simulation environment

## 6.1 GSL

## 6.2 GNUPlot

## 6.3 CERN ROOT

CHAPTER 7

# Donate