The PIPE Language: An Overview

Author: Pierre Menard
Email: Pierre.Menard@bms.com
Columbia ID: pm2146

Introduction

The earliest programmers implemented their logic in hardware. The barrier to entry was a PhD in electrical engineering and a soldering iron. Then machine language was invented which allowed more people to become programmers. By the time assembly language was created, a programmer only had to be slightly insane to be successful at his craft. Nowadays, modern languages are simple enough that anyone with an interest in computers can write a program. The evolution of programming is such that more people become programmers as languages get easier to use. However, some say there are more programmers today because they are needed to fix all the bugs in old programs. More programmers will be needed tomorrow to fix the bugs in today's programs.

Looking at the trends and the reasons behind them, any reasonable economist would immediately conclude that one day everyone will be a programmer. But before software developers around the world can rejoice over the inevitability of this logical argument, a programming language will be needed that even technophobes can use. That's where PIPE comes in. It's the general-purpose language that even grandma can master!

PIPE is an acronym for "**P**IPE **I**s **P**rogramming **E**volution". However, since PIPE hasn't proven itself yet, PIPE stands for "**P**IPE **I**s **P**ierre's **E**xperiment".

Conceptual Model

Many believe that the holy-grail of computing is Star Trek's on-board computer;

> Jean-Luc Menard - "Computer, open a hailing frequency to the Romulan warship."
>
> Computer - "Sir, the Romulans are ignoring your hail."
>
> Jean-Luc Menard - "In that case, fire photon torpedos at the hull"
>
> Computer - "Firing torpedos..."
>
> <explosions and earthquake-like vibrations>
>
> Jean-Luc Menard - "Computer, what happened?"
>
> Computer - "Photon torpedos were fired at the hull"
>
> Jean-Luc Menard - "Not our hull, you idiot! The Romulan's hull! Take us out of here! Warp 9!"

Most laymen (and software professionals) view automating a task as specifying the steps a program must take to reach the desired state. This is very much like procedural programming. PIPE doesn't deviate from this conceptual model, but does recognize that the amount of 'theory' required to learn procedural languages is too high for most people. Part of the problem is that programs are represented in text. It's easier for most humans to think in graphical terms. Rather than typing "Fire torpedos at the hull of the Romulan ship located at coordinates x,y,z", making sure to conform to the stringent syntax of a programming language, it is probably easier to drag and drop a picture of a torpedo onto the image of the target ship. While both are equivalent, the graphical procedure is preferred because it does away with an unnatural and exacting syntax.

The PIPE integrated development environment (IDE) is a modeling tool that graphically represents a computer program. For example, a PIPE programmer can drag and drop a 'loop' module into the current 'workflow'. Behind the scenes, the user's actions are translated into the PIPE programming language and eventually into Java. By design, the PIPE language is not very succinct. The reason is because PIPE syntax tries to mirror the feel of the graphical workflow model. Indeed, this is the justification for creating the PIPE programming language rather than implementing the workflow concept as an application in a standard language. Users who are curious about the underlying code can look at it (and modify it) and will have a better chance of understanding the semantics of the textual representation.

PIPE makes no attempts to be computationally efficient and doesn't care if it's a hundred times slower than an equivalent Java program. If a regular person can automate a task on their own, it's assumed a slow PIPE program will still be at least an order of magnitude faster than manually performing the steps by hand. Once a PIPE program has been proven useful, PIPE allows for a real programmer to implement the PIPE program in pure Java and expose that as a PIPE module. Java code can therefore be embedded into a PIPE program. In this way, PIPE users create the initial programs and software professionals can make them better and faster. Here's an example program that a high-school teacher might create;

Step 1: Drag and drop the 'open tab-delimited file' module into the workflow.
Answer the prompt that asks for a file name. (eg. grades.txt)
Step 2: Drag and drop the 'column iterator' module into the workflow.
Answer the prompt that asks for a column name. (eg. grade)
Step 3: Drag and drop the 'branch' module into the workflow and specify a branching condition.
The workflow now bifurcates. (eg. grade > 90)
Step 4: Connect a 'null' module to the 'false' port of the 'branch' module.
Step 5: Connect an 'export to Excel' module to the 'true' port of the 'branch' module.

This program opens and reads the grade column of a tab-delimited file and exports to Excel those lines where the grade is greater than 90.

Example Syntax

All PIPE programs begin with a START module. There can only be one START module per workflow. In the PIPE IDE, the START module is the canvas of the main working area. At the left of the work area, there are modules represented in a hierarchy that a person can drag and drop into the work area.

To implement Hello World, the PIPE programmer drags and drops the PRINT_SCREEN module onto the work area, and types in "Hello World" when prompted. Here is what the corresponding PIPE code would look like;

```
module START {
        call module PRINT_SCREEN {
                input @text is "Hello World".
        }
}
```

The START module invokes the PRINT_SCREEN module with "Hello World" as input to the PRINT_SCREEN module. All variables begin with the @ character. In the above example, @text is an input variable that is used by the PRINT_SCREEN module.

Now suppose that the Hello World program is so useful that we want to create a reusable module from it. We can save the workflow as a module and give it HELLO_WORLD as a name. Here is the PIPE code for that;

```
// user can create their own modules and share them with their friends
module HELLO_WORLD {
        call module PRINT_SCREEN {
                input @text is "Hello World".
        }
}
```

Here’s a variation of Hello World that illustrates how to handle outputs from a module;

```
module START {

        @answer is call module GET_USER_NAME { }

        call module PRINT_SCREEN {
                input @text is "Hello " + @answer->@name.
        }
}

module GET_USER_NAME {

        output @name.  // definition for outputs

        call module PRINT_SCREEN {
                input @text is "What is your name?".
        }

        // GET_USER_INPUT is a built-in module.
        @name is call module GET_USER_INPUT { }
}
```

The implementation of the built-in PRINT_SCREEN module is as follows;

```
module PRINT_SCREEN {
        input @text.     // definition of input

        call module JAVA {
                input @code is {
                        System.out.println(@text);
                }
        }
}
```

The PRINT_SCREEN module is implemented using the built-in embedded Java facility. PIPE comes bundled with several built-in modules that serve as a basis for the creation of useful programs. The PRINT_SCREEN module is just one of many such modules.