Script-Music-TV
Getting Started with *Tabula Vigilans*

by Dr Archer Endrich

˜A Richard Orton Legacy˜

> Tabula Vigilans ('TV') *has a very clear syntax and many built-in music-related functions that are not part of other programming languages, so we are proposing that it is a powerful ready-made music language that is ideal for a teaching environment. However, note that programming languages often have a lot in common. What we have written here can easily be 'translated' to another similar programming language (such as Python, SonicPi or whatever). There are usually just a number of little differences in syntax to remember. Thus there can also be a* Script-Music-Python, *for example, but with more to do to recreate TV's music functions.*
>
> Script-Music-TV *is a comprehensive introduction and tutorial manual set for* Tabula Vigilans. *It is based on the large number of scripts written by Richard Orton. These scripts comprise an important part of his legacy. In assembling them here, I have sought to use them both to show how to write scripts for 'TV' as well as to illustrate a number of musical objectives, goals placed very much more within reach than they would have been thanks to Richard's profound insight into the kind of processes useful to achieve effective musical results.* – Archer Endrich

About *Tabula Vigilans*

Tabula Vigilans ('TV') was conceived and written by **Richard Orton** (1940-2013) as a musical spreadsheet. The concept taken from that of a spreadsheet was the way it would recalculate all the values on the sheet at once. Richard realised how such a system could enable complex algorithmic instructions to be updated at high speed, thus making possible a real-time musical instrument, initially using MIDI (Musical Instrument Digital Interface).

The key to this instant recalculation and real-time operation was that the computer would loop around the main procedure of the program *at machine speed*. This procedure was named **start()** and it 'contains' all the other procedures of a script, whether within itself or called from it. Part of what one needs to learn in using TV is how to avoid everything happening at once!

TV is a specialised programming language for music. It has many built-in functions that are specially designed as musical facilities. Some of these are directly musical functions such as 'midiout' to play sounding output, and others are designed to produce and shape data for musical purposes. An example of the latter is 'lin', which can be used to produce time-varying output between defined limits. There are also a full range of mathematical functions.

A further development of *Tabula Vigilans* made it possible to run external programs from within a TV script. This was implemented by Professor John Ffitch and uses an 'system' function. (He is currently developing a keyword editor for TV.) This means that a TV script can be used to run external commandline programs, such as those in the Composers Desktop Project ('CDP') system. Furthermore, it can be used to write specialised breakpoint or text files used by the CDP programs. This opens up a vast field of possibility. It was Nick Fells who paved the way for this line of development by writing a TV script to output score files for *Csound*, and Archer Endrich later applied the method to CDP file formats when working on the LHCsound sonification project.

Tabula Vigilans runs from the command line. It helps to have the TV executable in your system path. A CDP user is recommended to put it in the CDP program directory (usually _cdprogs) and to ensure that this directory is in the system path. TV will then will run no matter which is your current directory.

On PC open cmd.exe (a shortcut on Desktop is helpful), and on a MAC open the Terminal. Then 'cd' (change directory) to the directory in which you want to work. Write your script with any plain text editor, bearing in mind that one which displays line numbers will be helpful when your scripts start to get longer. Give your script the extension '.tv'. If you use emacs, bear in mind that filenames (still?) are restricted to 8 characters plus the extension (which can be any two or three characters, a useful feature). You then run your script with cmd.exe or in the Terminal with command lines such as `tv ascript.tv`. If there should be sounding output, you can activate immediate playback with the `-i` flag: thus `tv -i ascript.tv`.

About Scripting

Scripting is your friend. We take graphic user interfaces (GUIs) to be the norm, so it can come as a bit of a shock that you are expected to type in text in order to run *Tabula Vigilans* and other scripting languages. This text is a 'script' *that is actually your own custom-designed computer program*. Not only that, but the resulting scripts are run by typing in a commandline in the command line interpreter (cmd.exe on PC and Terminal on MAC). This is why we have named this approach **Script-Music**. Despite the initial shock, it is a very, very powerful way of going about things. You are closer to the operating system of your computer, and you are very much in control of the open-ended evolution of your own algorithmic music system. **The ability to type fluently will increase your speed and enjoyment enormously**.

Besides the script, the interface with the computer is its command line interpreter. A command line has these components:

- the name of an executable program (with or without its extension)
- any parameters that the program requires, including any optional ones that you want to use
- the full name of the script you want to run, including its extension
- any parameters that the script requires, including any optional ones that you want to use

The 'usage' of a program gives its name and all required and optional parameters (ideally with explanations!). The usage is displayed if you just type in the program's name in the command line interpreter and then press 'Return'. 'Flags' are usually single letters preceded by a minus sign, and these invoke some function of the program. Optional parameters are shown inside square brackets, e.g., [-f]. Files may also be submitted to the program if it is designed to open and read them.

Here is a sample command line to run *Tabula Vigilans*:

```
   tv -i mytune.tv melody.txt
```

We invoke the 'tv' program with the '-i' flag for immediate playback, give it our handwritten script 'mytune.tv' and provide a text file which contains a series of pitches that mytune.tv will use. Thus we have what is really a very simple, direct, controllable, and extensible environment. What is not simple is music itself, and you will soon find that good musical results are not so easy to obtain, and programming expertise at some level is vital. But the lure of the possible is a powerful incentive, and the beauty of music is enticing. It is a very rewarding path. The built-in music functions of TV enable you to concentrate on data and control flow, and your innate musical imagination will be your guide.

About the start() Procedure

The `start()` procedure, around which the computer loops at machine speed, is where it all happens. This container for all the rest of your code looks like this:

```
start()
{
...
}
```

It will have some code within itself, and from it you will call other procedures to carry out specialised tasks. Designing and placing procedures is an essential part of learning to write computer programs effectively.

About Messages

NB: It is useful to put a shortcut to the *Tabula Vigilans* manual on your desktop! For example, here is the chart of keywords in TV: TV Keywords Chart

Tabula Vigilans, like any programming language, has a number of ways of writing messages, either to display on the screen or to write into a file to be saved externally. You will see these commands very early on in the example scripts, so it is just as well to learn about them now. Other programming languages will have similar commands. Here are the essentials.

- `message "Information to display on the screen\n"` – this will be shown every time the program 'sees' this command; note that the text is contained within double-quote marks. The `\n` places a line-break ('carriage-return') at the end; it may or may not be needed, depending on how you want the output to look.
- `messag1 "Read this only once.<\n>"` – This text will be displayed only once, no matter how many times the computer (loops round and) 'sees' it. It can be useful for diagnostic messages within loops.
- `probi integer` – `integer` is a variable that is ideally an integer, such as 7. If it is not an integer, only the integer (whole number) part of it will be displayed.
- `probe fp` – `fp` is a floating-point variable, i.e., with a whole and a fractional part, such as 7.52. Only two decimal places are shown, but you can
- `print bignumber, 10, 6` – the value of `bignumber` is for example 2.157042 and the print command here is telling the program to allow for a total of 10 characters (including the decimal point) and to display 6 decimal places.

A Template TV Script: template.tv

OK. We can finish 'Getting Started' with a template *Tabula Vigilans* script. There are 'comments' to tell what everything is. Comments in TV are begun with two foward slashes. They are neither run by the program or displayed to the screen, but when you look at your script, they help to remind you what it is meant to do. The usage for the script is displayed. We will also call an `initialise()` function and declare and fill a one-dimensional table from an external text file that is in the current directory. The script first displays the contents of the table as filled from the file and then displays the same numbers as calculated algorithmically (a cumulative addition). Finally, it allows you to specify your own numbers on the command line and will use those: one for the start number and one for the increment.

```
// Template TV Script: template.tv
// AE - 11 November 2013

table ATABLE[10]        //initialises a table of one dimension to hold 10 numbers
                        //note that it comes before start()
start()
{
        if(init == 0) {                 //fails if init == 1
                call initialise()               //go to the initialise() procedure
        }

        for(count = 0; count < 9; count += 1) { //do 10 times; NB use of 0 & 9
                number = ATABLE[count]  //reading table filled from ten_numbers.txt
                probi number            //display integer numbers
                message "\n"            //put a newline after each number
        }

        messag1 "\n"                    //put in a blank line
        messag1 "Create the same list of numbers algorithmically:\n"
        for(i = 0; i < 9; i += 1) {
                probi startnum          //start by displaying startnum (set to 9)
                message "\n"            //newline after each number
                startnum = startnum + 9 //add 9 to the previous number
        }

        messag1 "\n"                    //put in a blank line
        messag1 "Use the numbers you gave on your command line:\n"
        for(i = 0; i < 9; i += 1) {
                probi anynum            //start by displaying anynum (from arg(1))
                message "\n"            //newline after each number
                anynum = anynum + increment //add arg(2) to the previous number
        }
}

usage()
{
        message "Usage: tv scriptname.tv any_number increment\n" //generic usage
        message "\tany_number - your choice of number at which to begin the calculation\n"
        message "\tincrement - the number to add in a cumulative addition\n"
        message "Example: tv template.tv 10 5\n\n"       //this script
        //'message' is OK here because usage() is only called once
}

initialise()
{
        //display the Usage
        call usage()

        ATABLE fill_table "ten_numbers.txt"  //values in file read into the table

        startnum = 9            //hard-wiring this value
        anynum = arg(1)         //reads in first argument from the command line
        increment = arg(2)      //reads in second argument from the command line

        init = 1        //don't call initialise() again!
}
```

Last updated: 13 November 2013