Gerald's Column
by Gerald Fitton

This is the third tutorial article in the series which I hope will help you with PipeDream 4's custom functions. If you wish to write custom functions then you need to learn the PipeDream 4 Custom Function Programming Language. Custom functions written in PipeDream can be loaded into Fireworkz for RISC OS and Fireworkz for Windows.

Summary of Parts 1 and 2

Designing a new programming language is always a compromise between:

(a) Including commands which provide the flexibility needed to carry out complex or intricate processes quickly and ...

(b) Inherent constraints which encourage those 'good' programming techniques needed to facilitate debugging and program improvements.

The compromise is to learn a set of conventions (a sub set of the constraints of the language) which encourage 'good' programming and to aim to write 'well written' programs. Only after becoming master of the conventions should you break them knowing that you need the speed or efficiency which can not be achieved by sticking rigidly to those (advised) conventions.

A custom function is a sequence of commands which start with a '...function' command and end with a '...result' command. Once there is a custom function within a document then the whole document is a custom function document. Custom function documents behave differently from 'ordinary' documents (in particular, the sequence in which the slots are evaluated is different) - but they do not have a different file type!

Parameters are a 'sort of' variable which can be passed to custom functions. It is a good convention (not a constraint) that they retain their identity and value throughout the custom function. You can consider parameters to be variables which don't vary within the custom function.

The following conventions (not constraints) apply to the use of PipeDream Names within a custom function. Local variables should be declared as Names rather than using slots for local variables. It is a convention (not a constraint) to use the '...set_name("name",slotref)' command once only. Use it to assign a slot in the workspace to the local variable name. Thereafter, it is conventional to change the value of the local variable with the '...set_value(name,value)' command rather than with the '...set_name("name",value)' command. This convention aids debugging, modification and extension of the function.

High Technology Information Processing Systems

Some many years ago, when I had to give a series of introductory lectures on 'Those new computer things' (meaning desktop as opposed to mainframes) to skilled professionals who were 'non-computerate' (I hated that phrase), I would start by writing the heading of this paragraph on the blackboard (or OHP film - no Powerpoint in those days) as "HTIPS".

Then I would go on to explain the meanings of the words, emphasising that ‘digital electronics’ were used for coding, storing and processing information. I went on to lead a discussion (‘student centred learning’ was in vogue) of the advantages and disadvantages of ‘electronic’ processing. Part way into the first lecture I would suggest that a useful way of classifying such ‘Systems’ was in terms of the way in which the data (information) was created and processed.

The first category I called “System created” and it consisted essentially of minimal input and maximum processing by the ‘System’. Because data storage (memory) was limited on early computers they tended to be of the “System created” type. A good example is a prime number generator. Another example is a program in which you enter a year (for which the program is guaranteed to give valid answers) and the output is the date of Easter Sunday for that year.

The second category I called “User created”. Examples are simple text editors and simple ‘store and retrieve’ databases. Early programs of this type used the ‘IP System’ for ‘electronic’ storage with little if any processing. These days our data might be not only words but photographs, audio files and (perhaps not on a RISC OS machine) even videos.

The third category I called “Expert created”. Some of the earlier “Expert created IP Systems” included telephone directories and the Bible where the ‘System’ was used mainly as a rapid access store.

Data Validation

Let us not worry too much about classification. What is important is that whatever data you (or anyone else) enters into such a system must be data which is within the range for which the ‘System’ gives valid answers. This brings me round to ‘Data Validation’ and ‘Conditional Execution’. Essentially you want your custom function to be executed if and only if it gives the correct answer for the data which has been entered.

For example, in the UK the way in which leap years are calculated changed in the year 1752 so, if the custom function you write gives correct answers only for years after 1752 then you must ‘test’ whether the year input is before or after 1752. What I have rather loosely called a ‘good’ program would test the data entered and execute the custom function only if the result is valid.

A rather more sophisticated custom function for determining if a year is a leap year could ‘work’ for dates from 5AD onwards (8AD was the first leap year). Such a custom function would use one formula for years after 1752 and a different formula for years between 5AD and 1752. In 1752, in the UK, the rule for determining a leap year changed.

If you wished to make the custom function even more ‘sophisticated’ then you could take into account the fact that some countries changed their calendar not in 1752 but in some earlier or later year! For such a custom function to return the ‘correct’ value you would have to enter not only the year but also the country.

What I am getting round to is that a most useful command in any programming language is one which tests a condition and, if the condition is true then one sequence of commands is used, whereas, if the condition is false another sequence has to be used.

In a PipeDream custom function there is a command with the syntax:

...if(condition,do_if_true,do_if_false)

The condition often involves testing if one variable is equal, greater than or less than another; it can always be framed as a question (eg “Has ‘Fred’ more than 85% in his maths exam?”). If the answer to the question is “Yes” then the condition is true and the command I have called ‘do_if_true’ is executed; if the answer is “No” then ‘do_if_false’ is executed.

It’s time to look at an example. The files to which I refer are available from many places from the Archive monthly disc to the Archive website. If you have problems getting hold of them then please email me and I’ll get them to you!

Double click on the file [StuGrade] and you’ll find that it and a custom function document called [c_Grading] are loaded. What the first custom function (called from [StuGrade] column D) does is to award a grade (‘Distinction’, ‘Merit’, ‘Pass’ or ‘Fail’) according to the mark (between 0% and 100%) shown in [StuGrade] column B. The mark for student Fred Bloggs is in [Stugrade]B4 and, with 60%, he is awarded a Pass. Try a few more values between 0% and 100% (ignore for now the comments in the E column) and you will find Fred’s grade changes at 40%, 65% and 85%.

Where does the data validation come in? Well, if the student has a negative mark or one over 100% then it is likely that a typing error was made during data entry into StuGrade]B4. In either of these cases you will not want a grade to be calculated, you will want an error message to be returned instead. Try giving Fred 1000% or -10% and see what is returned in slot [StuGrade]D4.

Conditional Execution

Now let’s have a look at the custom function ‘...[c_Grading]grade’ and we’ll see how the input data is validated.

The two lines which are used to validate the input data are [c_Grading]A15 and [c_Grading]A16. The command used is the ‘...if(,,)’ command. In [c_Grading]A15 the condition can be phrased as the question “Is the parameter @marks greater than 100?” (Note that @marks needs an @ sign in front of it since it has been passed as a parameter to the function and it is not a name - see earlier articles).

If the condition is true (ie if the answer to the question is “Yes”) then you don’t want to find the corresponding grade but you want to return an error message instead. One simple way of implementing this is to return from the custom function immediately by using the command ‘...result(“Over 100?”)’ as I have done here.

If the condition is false (the answer to the question is “No”) then you want to find the grade (using [c_Grading]C20 to C25) corresponding to @marks. If you look at the command in [c_Grading]C15 you will find.

...if(@marks>100,result(“Over 100%”?),)

I want you to notice that the last two characters are ,) so there is no ‘do_if_false’ command. The effect of there being no such command is that the “Sequence” (see the earlier tutorials) of commands which are executed continues down column A of the custom function. In this case the ‘program pointer’ moves down to [c_Grading]A16 and tests for a negative @marks in a similar way.

Using an Inequality

If @marks is not greater than 100 and not less than 0 then it must lie between 0 and 100 inclusive! Lines [c_Grading]A20 to A23 are all of the same type as those we have just studied but let’s have a look at what happens in detail if @marks has the value 60.

As recommended in the earlier tutorials, at the beginning of the custom function, in slot [c_Grading]A10 I have declared the variable name “grade_awarded” allocating to it slot [c_Grading]B10. After using ‘...set_name(“name”,slotref)’ once (and once only) this way I use the command ‘...set_value(name,value)’ in [c_Grading]A20 to A23 to change its value.

Looking first at [c_Grading]A20, if @marks = 60 then it is certainly less than 100 so the condition is true (a “Yes”). Hence a “Distinction” is (provisionally) awarded to our hero Fred. His elation (if he could read quickly enough) would be short lived because the ‘program pointer’ moves on to the next slot and, in the same way, awards Fred a “Merit”. the condition in [c_Grading]A22 is also true and so Fred gets a “Pass”! If you look at the next line, the condition asks the question “Is @marks<40?“ to which the answer is ”No“ and Fred’s ”Pass“ is upheld. He is not awarded a “Fail” because the command ‘...set_value(grade_awarded,“Fail”)’ is not executed.

Only the final value of the name “grade_awarded” is returned from slot [c_Grading]A25 to slot [StuGrade]D4 so that, whilst [c_Grading]B10 has changed from its original value to “Distinction”, then to “Merit” and finally to “Pass”, the value in [StuGrade]D4 is only ‘updated’ when the ‘program pointer’ has passed from [c_Grading]A25 back to [StuGrade]D4.

Using an Equality

Rather than using the less than inequality (eg @marks<100), in the next custom function I have used equalities. This next custom function is called from [StuGrade]E4 as ‘...[c_Grading]first_comment(C4)’ with one parameter, the value held in slot [StuGrade]C4. Note that the value in [StuGrade]C4 is text and not a number. I have used letters such as A, B, C, D, E, F, G as codes instead of the characters (by which I do not mean numbers) 0, 1, 2, 3, 4, 5 and 6.

I would like you note how the data is validated. In [c_Grading]A35 I have assigned a default value to the variable “comment_01”. All the ‘...if(,,)’ lines of [c_Grading]A37 to A43 are executed because I have not used ‘...result()’ as I did in [c_Grading]A15.

However, if the data is invalid (not one of the characters A to G) then the conditions of these lines will be false and so ‘...set_value(comment_01,value)’ will not be executed. The ‘...result(comment_01)’ of [c_Grading]A45 will be the default value.

Using a ...vlookup(,,) table

My third custom function, [c_Grading]second_comment(C4), is called from [StuGrade]E5. It appears in [c_Grading]A48 to A68. Once again, my strategy for data validation is to set an 'error message' default value for the variable which is returned by the custom function. If the character in the ASCII code is lower than "A" or higher than "G" then the vlookup(key,range1,range2) function will return "Invalid".

I would like you to notice that the columns of the lookup table are given Names in [c_Grading]A51 and A52 and that these Names are used in slot [c_Grading]A57 where there appears:

...set_value(comment_02,vlookup(@code,code_table,comment_table))

Also, I recommend the convention that the "Expert data" contained in the lookup table is placed to the right of column B and below the line containing the '...result()' command.

The advantage of doing this is that you can add or delete rows from the custom function with impunity. If you had data in the same row as a custom function command then adding new rows would make the data columns 'ragged'. If you delete such a row then you will lose data.

As an alternative to including 'Expert data' in the custom function when it is much larger than the databases of this example, then you should consider storing the data in a separate dependent document. This strategy has the advantage that the database can be changed without changing the custom function file.

Over to you

Generally, unless the "Expert data" is limited to two or three values, I recommend to you that you use a vlookup table as I have done in my third custom function rather than using the multiple '...if(,,)' commands of the first two custom functions. Such vlookup tables are much more easily modified than conditions in '...if(,,)' commands. Furthermore, the lookup table can be searched and sorted.

What I suggest you do is to rewrite the second custom function so that it uses a vlookup table rather than the multiple '...if(,,)' commands.

When you have done that successfully then try to rewrite the first function in terms of a table. This is much harder because the first function uses inequalities. The table must be sorted in ascending order of @marks and, if an exact match is not found then vlookup(,,) will return the grade corresponding to the next lower value in the @marks column. If you have insuperable problems then send me an email and I will provide you with my vlookup(,,) solution.

Summary

Commands in a custom function can be conditionally executed.

A command used for this is the '...if(,,)' command. One of its principal uses is the validation of data passed to custom functions as parameters. Indeed, in larger programs, it is often the case that data validation routines use more code than the 'wanted' processing!

Lookup tables are preferable to multiple '...if(,,)' statements unless the number of 'options' are few

In a later tutorial on 'sub routines' we shall see that '...if(,,)' can be used to make substantial changes in the sequence in which commands are executed by calling one sequence if the condition is true and a completely different sequence if it is false

The usual way of 'Conditional Branching' to a sub routine is to use

'...if(condition,one_routine,alternative_routine)' as the command

Communications

Please write to me at archive@abacusline.co.uk with your comments.