Ms Resi Development Manual – Edition 2 – February 2009

(web links updated June 2012)

Mal Haysom
m.haysom@cartography.id.au

Abstract

This manual contains information to assist developers to modify the physiological component of the Ms Resi code. Ms Resi consists of two modules – the interface module and the model module – the latter contains the core of the physiological code. To modify this module a developer will require a command line C++ compiler, a text editor, some knowledge of the C programming language, and a more substantial knowledge of physiology. The manual provides explanations and example modifications to guide developers. It provides contact details for a suitable (and free) compiler and text editor.

Contents

1. Introduction 1
1.1 Notation 1

2. A walk through example 2

3. Files that are used to build the model module 3
3.1 `ms_model_100.cpp` 3
3.2 `ms_variable.h` 3
3.3 `ms_subject.h` 3
3.4 `makefile.mak` 3

4. Making more significant changes 4
4.1 Inserting another display variable 4
4.2 Rearranging the subject factors and display variables 4
4.3 Changing the subject constants 4
4.4 Changing the model algorithms 4

5. About trade 5

6. Managing multiple versions 6

7. Coding Information 7
7.1 General 7
7.2 Deviations from Appendix IV 7
7.3 Notation 7
7.4 Tracheal obstruction and bag code 8
7.5 Respiration dialogue box funtionality 8

8. C++ syntax used in building model module 10
8.1 The language 10
8.2 The build process 10

9. References 11

Appendix A – The MS Windows command line syntax 12

Appendix B – The damp function 13

1. Introduction

The Ms Resi application code consists of two modules: `ms_resi_200.exe`, a module that contains the graphical interface code and `ms_model.dll`, a module that contains the physiological code. The graphical interface module has been built using Borland Builder 6. The physiological core module can be built from the supplied source code, `ms_model_100.cpp` by using a free C++ compiler. This arrangement will allow researchers, perhaps with a particular area of interest, to develop the physiological code without being involved in the internals of the graphical interface code.

On the whole, the core C++ code that is used to build the model module is a direct translation of the Fortran code given in Appendix IV of Dickinson's work [1]. It is strongly recommended that a developer has this work to hand.

The designers of the Appendix IV model are Dr. C. J. Dickinson, Dr. E. J .M. Campbell, Dr. A. S. Rebuck, Dr. N. L. Jones, Dr. D. Ingram and Dr. K. Ahmed.

In 1981-2 Professor George Havenith [5], then at the Theoretical Biology Group, State University, Utrecht made some refinements to the Appendix IV code. He has made two reports on his work and the modified code is available on the Chime Contributor's web site [6]. These items also are strongly recommended to developers. The changes made by Havenith will be termed GH code.

Because the files that make the interface module and the files that make the model module can be independently modified, each module has its own version number – currently 2.00 and 1.00 respectively. Both version numbers are displayed at the start-up of Ms Resi.

1.1 Notation

Within this manual file names, code extracts and command line entries are presented in `Courier New` font. MS Windows menu paths and screen messages are printed in *italics*.

2. A walk through example

As a demonstration of the process, let's make a minor change to the model.

1. Download the C++ compiler and linker from Embarcadero Technologies [2]. Read the *Supplementary Information* available from Embarcadero Technologies [3].

2. Install the compiler and linker. (The configuration files are available, see step 4.)

4. Make a `ms_resi` directory
Unzip `ms_resi_200.zip` and `ms_model_100.zip` into this directory. The directory contents should include:

`makefile.mak`	
`ms_model.dll`	
`ms_model_100.cpp`	
`ms_resi_200.exe`	
`ms_variable.h`	
`ms_subject.h`	
`borlndmm.dll`	(Borland redistributable)
`cc3260mt.dll`	(Borland redistributable)
`cc3260.dll`	(Borland redistributable)
`rtl60.bpl`	(Borland redistributable)
`stlpmt45.dll`	(Borland redistributable)
`vcl60.bpl`	(Borland redistributable)
`bcc32.cfg`	(Compiler configuration file, move to C:\borland\bcc55\bin)
`ilink32.cfg`	(Linker configuration file, move to C:\borland\bcc55\bin)

5. Make backup copies of `ms_model_100.cpp` and `ms_model.dll`.

6. With a text editor edit `standard version` in line 30 of `ms_model_100.cpp` to `modified by your name`. (The Bloodshed Software compiler [4] has a useful syntax coded editor. However, I have not been able to use this compiler to build `ms_model.dll`.)

7. In *Programs->Accessories->Command Prompt* find your way to your `ms_resi` directory.

8. At the command line prompt enter `make`. The exchange should be similar to :

```
C:\ms_resi> make
MAKE Version 5.2  Copyright (c) 1987, 2000 Borland
 bcc32 -c -tWD -tWM- -X- -r- -a8 -k ms_model_100.cpp
Borland C++ 5.5.1 for Win32 Copyright (c) 1993, 2000 Borland
MS_MODEL_100.CPP:
 ilink32 /Tpd /w /aa /Gi /Gn c0d32.obj ms_model_100.obj, ms_model.dll,,import32.lib cw32i.lib
Turbo Incremental Link 5.00 Copyright (c) 1997, 2000 Borland
```

9. Run `ms_resi_200.exe`.

10. In Ms Resi key *Alt-I* ("I" for Information). A dialog box displays *modified by your name*.

The process described in the box above has produced a new `ms_model.dll`. When `ms_resi_200` ran, she accessed this new module, and thus the *modified by your name* string was available for display. In section 4, methods for making more significant changes to the model are presented.

3. Files that are used to build the model module

Several files are required to build the module model.

3.1 `ms_model_100.cpp`

This file contains code that constitutes the physiological core of the model. It also declares the passport parameters, the subject factors, the display variables and other variables. The passport parameters (with the exception of `sex`), the subject factors and the display variables use the structure `variable` declared in `ms_variable.h`.

```
// --- variable structure ---------------------------------------------------
struct variable {
                char *dsn;                    // text description of variable
                float vlu;                       // the value of the variable
                float lwr; // lower limit for user entries or graphical display
                float upr; // upper limit for user entries or graphical display
                char *unt;                               // text description of unit
                };
```

For example, in the subject factors array we find:

```
{"Packed cell volume",                              0.0,  0.0,  80, "%"},
  #define PCV   factor[21].vlu  // (19)
```

The entry above declares a user input variable `Packed cell volume`. The initial value (changed at the initialization of the subject) of this variable is `0.0`, the lowest value of this variable that Ms Resi will accept is `0.0`, the highest value is `80`, and the units are `%`. `PVC` is defined as a macro substitute for the actual expression for the value of this variable, `factor[21].vlu`. The `(19)` is the variable number in the Appendix IV code.

In the list of physiological variables array we find:

```
{"Arterial pH",                                     0.0, 7.0,  8.0, ""},
  #define RPH   display[17].vlu     // (33)
```

The entry above declares a display variable `Arterial pH`. The initial (soon to be changed) value of this variable is `0.0`, the lowest ordinate value of this display variable is `7.0`, the highest ordinate value is `8.0` and the variable has no units. `RPH` is defined as a macro substitute for the actual expression for the value of this variable – `display[17].vlu`. The `(33)` is the variable number in the Appendix IV code.

The `other` variables are required for the model, but not for user control or display. They are variables internal to the model.

3.2 `ms_variable.h`

This file contains declarations common to both the interface and model modules. The expectation is that it will not be altered by a developer.

3.3 `ms_subject.h`

This file contains the clinical codes used by the function that generates symptom messages. It also contains the wherewithal to initiate the default subject.

3.4 `makefile.mak`

This file contains the instructions that the compiler and linker require to build `ms_model.dll.`

```
ms_model_100.obj: ms_model_100.cpp
 bcc32 -c -tWD -tWM- -X- -r- -a8 -k ms_model_100.cpp
 ilink32 /Tpd /w /aa /Gi /Gn c0d32.obj ms_model_100.obj, ms_model.dll,,import32.lib cw32i.lib
```

4. Making more significant changes

Since all the physiological code and the bulk of the physiological variables reside in the files that are used to build the model module, there is considerable scope for a developer to make changes to the model.

4.1 Inserting another display variable

This example, although perhaps physiologically meaningless, (I'm an instrumentation engineer ;-) will illustrate some aspects of the interaction between the two modules.

The aim is to display a new variable – the averaged pH of the arterial, tissue and venous pH values.

Step 1 – Declare the variable – circa line 286 in `ms_model_100.cpp`, the new code is in bold font:

```
{"Venous pH (of mixed venous blood)",                0.0, 7.0,  8.0, ""},
  #define VPH   display[62].vlu     // (34)
{"Arterial, Tissue and Venous average pH",           0.0, 6.6,  7.6, ""},
  #define RTVPH   display[63].vlu   // (new)
{"No display",                                         0,   0,    0, ""}
```

Note the array index of `63`; it is an increment of 1 from the previous display index. We have also elected that the display plot will range from `6.6` to `7.6`.

Step 2 – Assign a value to `RTVPH` (the symbol allocated to our new variable) in the model – circa line 866 in `ms_model_100.cpp`, the new code is in bold font:

```
M_END: RTVPH = (RPH+TPH+VPH)/3.0; // determine average pH before return to interface module
       return(item);
```

Step 3 – In the command line window run `make` to build the revised `ms_model.dll`. Run `ms_resi_200.exe` and observe the presence of the new display variable, *Arterial, Tissue and Venous average pH*, at the bottom of the display list. Observe that when this variable is selected the ordinate axis ranges from *6.6* to *7.6*.

4.2 Rearranging the subject factors and display variables

In the example above, the display variable, *Arterial, Tissue and Venous average pH*, appears at the bottom of the display list. To be consistent with the alpa-numerical ordering it should be placed higher up the list, after *Arterial pool O2 content of blood leaving*. The order of the variables can be rearranged subject to two requirements.

1) For the display variables list, `No display`, must be the last item in the list.

2) The indexes in all the `#define` statements, for example, the "`62`" in

```
#define VPH display[62].vlu,
```

must be readjusted to maintain numerical order. For variables with alternate units, for example,

```
#define SN2PR display[48].vlu // (105)
#define AD_9 48
```

the correction to the original index (here `48`) must be done on both define lines.

In addition, the order of the subject factors and display variables as they appear in `ms_subject.h` must be rearranged for consistency.

4.3 Changing the subject constants

The subject constants are defined in the function `constant()` (line 1108 in `ms_model_100.cpp`). These constants are evaluated before each run of the model. For a developer they provide a painless means of adjusting some parameters of the model. For example, `C[75]`, defined as

```
C[75]=(1.5+s_sex*0.2+(33.0-FITNS)/10.0)*WT*C[16]; // maximum stroke volume GH
```

relates heart stroke volume to sex, weight and (via `C[16]`) cardiac functionality. If a researcher was working with a subject group where the cardiac capacity of its members was believed to be independent of their sex, the term "`s_sex*0.2`"could be removed.

4.4 Changing the model algorithms

Perhaps the reader could examine the changes made by Havenith [6] to gain an appreciation of the scope available for changing the actual model.

5. About trade

"I run a sort of import-export business; I guess you'd call it. Anything people can't get, I can."
Harry Smith (Humphrey Bogart) *Sirocco* 1951

Some declarations in the source files for `ms_model.dll` have a `trade` specifier.

```
trade int model(void);             // Dickinson's physiological model (in ms_variable.h)
trade variable height = {"height", 0.0, 95.0, 200.0, "cm"};     // (in ms_model_100.cpp)
```

The specifier `trade` is defined in `ms_variable.h`.

```
#ifdef __DLL__
       #define trade extern "C" __declspec(dllexport)
#else
       #define trade extern "C" __declspec(dllimport)
#endif
```

The result of this conditional definition is that the model-related files interpret `trade` as an export specifier and the interface files interpret `trade` as an import specifier. This import-export business is required to achieve linkage between the interface and model modules. Interference with the `trade` declarations will result in linkage problems at run time.

6. Managing multiple versions

Developers may wish to have several concurrent versions of the model. With appropriate accommodations, file names, with the exception of `ms_model.dll`, can be changed to reflect different versions. The interface module expects to link to `ms_model.dll` and will be distressed if it is not available.

A straightforward way of managing different versions would be to create separate directories each with its own copy of `ms_resi_200.exe`. Each Windows operating system has a search path algorithm for locating the DLLs required by any application program. Typically this search path includes `C:\Windows\System\` (or `C:\WinNT\System\`). Hence some duplication of files could be avoided by putting the Borland distributables (`borlndmm.dll, cc3260mt.dll, rtl60.bpl, stlpmt45.dll, vcl60.bpl`) in the appropriate directory where `ms_resi_200.exe` will find them as required.

The main source file, `ms_model_100.cpp`, may be renamed provided the appropriate changes (4 places) are made in `makefile.mak`.

7. Coding Information

7.1 General

On the whole the `ms_model_100.cpp` C++ code is a direct translation from the Fortran code given in Appendix IV of Dickinson's work [1]. It is strongly recommended that a developer has this work together with Havenith's modifications [5] to hand.

The quality of the translation varies (especially for conditional branches) as I became more familiar with the Fortran syntax. The original text was scanned using optical character recognition – errors occurred. Some of these errors I may have missed and some errors I may have introduced in the translation process. I am increasingly confident that transcription and translation errors in the core of the model have been eliminated, however checking against the original code is advised.

7.2 Deviations from Appendix IV

Significant coding changes that have been made are itemized below.

7.2.1 The iteration period is fixed at 1 second (the Appendix IV code iteration period is variable from 2 to 10 seconds). One second is a convenient period for the graphical display process and is laughed at by modern computers.

7.2.2 Within the core code, the pressure units are not selectable, but fixed as *mmHg*. This arrangement is more convenient for Ms Resi who prefers to make her own changes. As a consequence `SIMLT` is not present in `ms_model_100.cpp()`.

7.2.3 The bulk of the main iteration loop code (that is the core of the model) is in the function, `model()`, in `ms_model_100.cpp`.

7.2.4 The `delay()` function uses a different technique than the one that was used in the Appendix IV code. The principal change is that the current version uses a look-up table, `dly_dt`, and an interpolation function, `interpol()`, to determine the delay period in place of the algorithm used in Appendix IV. The look-up table approach will permit developers to make changes to the delay/cardiac output relationship more readily. The `interpol()` function was introduced into MacPuf by Havenith for other applications in the code.

7.2.5 There are extensive changes to the I/0 code.

7.2.6 There are extensive changes to the code associated with tracheal obstruction and the collection and rebreathing of expired gases in bags. Please refer to section 7.4.

7.2.7 Havenith's physiological code changes have been incorporated (including the introduction of work load and heart rate).

7.2.8 The code associated with the damp function/macro has been changed. Please refer to Appendix B.

7.3 Notation

7.3.1 Appendix IV page numbers are shown – `// P 198`. Code due to Havenith is noted `GH`. Code changes arising from the revamped `damp()` function are noted `TC`.

7.3.2 Labels (where required) have been constructed from the Fortran line numbers by prefixing with (usually appropriate) alpha characters. For example line 130 in the Appendix IV gases function is labelled `G130:` in `ms_model_100.cpp`.

7.3.3 Symbols for the major variables such as `PC2CT, TC2CT, DVENT`, etc, etc have been preserved.

7.3.4 At the user interface the Appendix IV factor numbers are preserved. For example, factor 2 is *Inspired* CO_2 in both the current code and the Appendix IV model.

7.4 Tracheal obstruction and bag code

In chapter 20 of *A Computer Model of Human Respiration*, [1] Dickinson discusses the code associated with tracheal obstruction and the collection and rebreathing of expired gases in bags. As a result of the necessarily strong interaction between the user interface and the respiration code, the MS Resi implementation of this code differs markedly from the Appendix IV version, though the function `bager()` remains as a sort of collection bag for an assortment of relevant code.

User data entry in Ms Resi is performed by the respiration dialogue box, accessed via the *respiration* item on the main menu bar. The display of the associated variables (not fully implemented) is done by the interface module in the same manner as it for the physiological variables.

Dickinson presents five respiration states.

1. Closure of the glottis;
2. Collection of expired air in a bag;
3. Rebreathing from a bag;
4. Rebreathing from a bag with CO_2 absorber attached;
5. Restoration of the *status quo*: glottis open, bag disconnected.

In Appendix IV code these states are numerically coded in such a way as to allow the economies of coding that were necessary in 1977. In Ms Resi code the states are macro defined in `ms_variable.h` as presented below. The macros `GTC`, `STD`, etc are the values that `PL`, the respiration index, can acquire.

```
// --- bag states --------------------------------
#define GTC 9                    // glottis closed
#define STD 10               // standard situation
#define CXA 11 // collection of expired air in bag
#define RBB 12           // rebreathing from a bag
#define CDA 13 // carbon dioxide absorber attached
```

The function `bager()` is declared in `ms_variable.h`..

```
trade void bager(int N, float *CA, float *CB); // deals with bag rebreathing
```

The first parameter, `int N`, is a tag that used by `bager()` to determine the origin of the current call. The function `bager()` may be called from two places in the interface-module code. These origins are defined by macros in `ms_variable.h`.

```
// --- interface module bager() origins -----------------------
#define RUN 1                                // run button click
#define RSP 2                        // respiration dialogue box
```

The function `bager()` may be called also from places within the model-module code. These origins are defined by macros in `ms_model.h`.

```
// --- local bager() origins ----------------------------------
#define MM1 11                   // call from model() circa M781
#define MM2 12                   // call from model() circa M820
#define MM3 13                                  // not allocated
#define MM4 14                                  // not allocated
#define MM5 15                                  // not allocated
#define MM6 16                                  // not allocated
```

7.5 Respiration dialogue box functionality

On acceptance by the *OK* button the respiration dialogue box code will set `NARTI`, `RRATE`, `TIDVL`, `PEEP`, `PL`, `BAGV`, `BAGC` and `BAGO` as appropriate and call `bager(RSP, NULL, NULL)` before closing.

My current implementation of the respiration code is incomplete and faulty. However, the process is sufficiently intact to illustrate its intended operation.

For example, assume that the user has elected to close the glottis. When the *OK* button is clicked in the respiration dialogue box, `PL` will be set to `GTC` and `bager()` will be called before dialogue box is closed. The switch code in `bager()` will recognize the origin of the call (via the `RSP` tag in the call) and that the glottis has been closed (as `PL` is set to `GTC`). On the next user *run* click the change in `PL` and those made by the `bager()` code will be effective.

```
case RSP:                                          // from respiration dialogue
     if(PL == GTC)                                         // if glottis closed
         REFLV = VLUNG;                                 // remember lung volume
     if((PL==STD)&&(ppl==GTC)) // if glottis has just been opened, breathe air
         {
         VLUNG = REFLV;                                  // restore all to normal
         FIO2 = 20.93;                        // bag stays filled as it was left
         FIC2 = 0.03;
         }
     // other RSP situations to be placed here
     break;
```

When the user elects to reopen the glottis by selecting *standard conditions*, the resulting call to `bager()` that occurs when the respiration dialogue box is closed, will again result in program flow through the `RSP` case. However, the new state of `PL` (now `STD`), together the previous state of `PL` (maintained by `bager()` as `ppl`) will result in the appropriate program flow within the `RSP` case.

8. C++ syntax used in building model module

8.1 The language

The C programming language was developed by Dennis Ritchie and Brian Kernighan. They published their classic work on C in 1978 [7]. The language was formalized as ANSI C in the late eighties. The object orientated language C++ ("C with classes") was developed by Bjarne Stroustrup over the period 1983-1985. C++ is a superset of C.

Although managed by a C++ compiler, the code used to produce `ms_model.dll` does not involve classes and is essentially traditional C. A developer unfamiliar with the C++ syntax might be well served by using one of the many books available on C (rather than C++) as a reference.

8.2 The build process

Traditionally *C source code* is *compiled* to produce *object code*. In the compilation process *header files* may be *included* in the source files. Object files produced from the source code are *linked* with library files to produce *executable* code. With more sophisticated operating systems, the concept of dynamic linked libraries, DLLs, was introduced. Dynamic link libraries (`ms_model.dll` is one) are linked to the application program (`ms_resi_200.exe` is one) at run time. There are advantages to be gained in memory usage and maintenance flexibility by this arrangement.

The DLLs can be produced by much the same process as described in the first two sentences of 8.2.

9. References

[1] *A Computer Model of Human Respiration*, C. J. Dickinson, MT Press, 1977

[2] Embarcadero Technologies downloads, https://downloads.embarcadero.com/free/c_builder (accessed 03/061/2012)

[3] Embarcadero Technologies, Supplementary Information (step by step installation) http://edn.embarcadero.com/article/21205 (accessed 03/06/2012)

[4] *Bloodshed Software Dev-C++ 5* http://www.bloodshed.net/ (accessed 03/06/2012)

[5] Loughborough University, http://www.lboro.ac.uk/departments/lds/staff/professor-george-havenith.html (accessed 03/06/2012)

[6] It appears that Mac Models material is no longer available at UCL CHIME, so I have posted the documents on this site. (Well, after all they are thirty years old.) Havenith report part 1, Havenith report part 2, MACPUF 17 December 1981 Fortran code.

[7] Kernighan, B and Ritchie, D, *The C Programming Language*, Prentice Hall, 1978

Appendix A – The MS Windows command line syntax

The Command Prompt program (*Programs->Accessories->Command Prompt*) was introduced to support MS-DOS applications that do not run under MS Windows. In essence it is MS-DOS without the short file name format limitations. However, since the bulk of the commands can be more readily implemented in the MS Windows environment, a model module developer need be familiar with only a few commands to operate effectively.

To change the drive, enter the drive letter followed by a colon (user entry in bold font).

```
C:\Documents and Settings\ADeveloper>Z:
Z:\>
```

To change the working directory, enter the path.

```
Z:\>cd ms_resi\ms_dallas
Z:\ms_resi\ms_dallas>
```

A back slash changes the working directory to the base directory.

```
Z:\ms_resi\ms_dallas>cd\
Z:\>
```

Enter the program name to run the program.

```
C:\ms_resi> make
MAKE Version 5.2  Copyright (c) 1987, 2000 Borland
     bcc32 -c -tWD -tWM- -X- -r- -a8 -k ms_model_100.cpp
- - - - - - - - - - - - - - - - - - - - - - - - - -
Turbo Incremental Link 5.00 Copyright (c) 1997, 2000 Borland
C:\ms_resi>
```

The Command Prompt interface has some useful features.

The up and down arrow keys allow a user to scroll through past commands.

To copy text from the display, use the menu via the icon in the top left corner, *edit* -> *mark*. Mark the text by holding the right mouse key and sweeping. The enter key copies the text to the clip board.

The output of commands can be piped to a file. For example, to make a file of directory contents.

```
Z:\ms_resi\ms_dallas>dir > list.txt
```

We can view the list.

```
 Z:\ms_resi\ms_dallas>type list.txt
 Volume in drive Z is Work Disc
 Volume Serial Number is 3865-1732
 Directory of Z:\ms_resi\ms_dallas
15/11/2009  11:37       <DIR>          .
15/11/2009  11:37       <DIR>          ..
15/11/2009  11:37                    0 list.txt
15/10/2009  14:55                  718 makefile.mak
10/11/2009  22:24               69,628 ms_model.cpp
10/11/2009  22:24              640,000 ms_model.dll
 - - - - - - - - - - - - - - - - - - - - - - - - - -
10/11/2009  22:24                3,732 ms_model.lib
10/11/2009  22:24              657,991 ms_model.obj
10/11/2009  22:23                4,279 ms_resi.dsk
10/11/2009  22:22              284,672 ms_resi.exe
             41 File(s)      4,889,782 bytes
               2 Dir(s)  18,109,349,888 bytes free
```

A help system is available.

```
Z:\ms_resi\ms_dallas>help
```

Appendix B – The damp function

The damp function serves two purposes in the model code. It models the mixing of fluids within a pool and is used as a stabilizing tool.

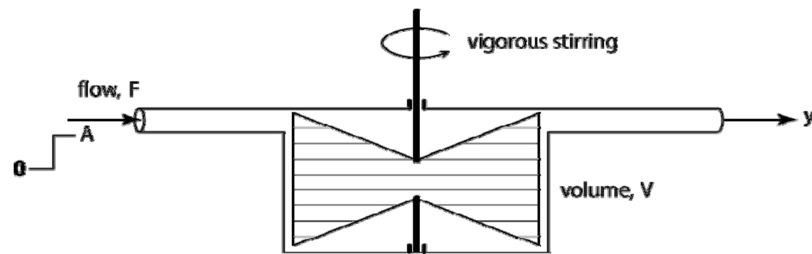

Fig 1.

A simple model of concentration changes within a fluid pool is shown in Fig 1. A step change in concentration from zero to A has occurred at the input at $t = 0$.

In an incremental time, dt, the amount of concentrate entering the pool will be $AFdt$; and amount exiting will be $yFdt$. Thus the change in concentration in the incremental time will be:

$$dy = \frac{Fdt(A - y)}{V} \qquad \text{equ. 1}$$

that is $$y + \frac{V}{F}\frac{dy}{dt} - A = 0 \qquad \text{equ. 2}$$

assuming the initial concentration of the pool to be zero then the differential equation, equ 2, has the solution:

$$y = A(1 - e^{\frac{-t}{\tau}}) \qquad \text{equ. 3}$$

where $\tau = V/F$. This parameter is known in some disciplines as the time constant of a single pole low pass filter.

The response of a low pass filter in the time domain to a step change in input is shown in Fig 2. After a period of one time constant the output has reached 63% of the span to the final value.

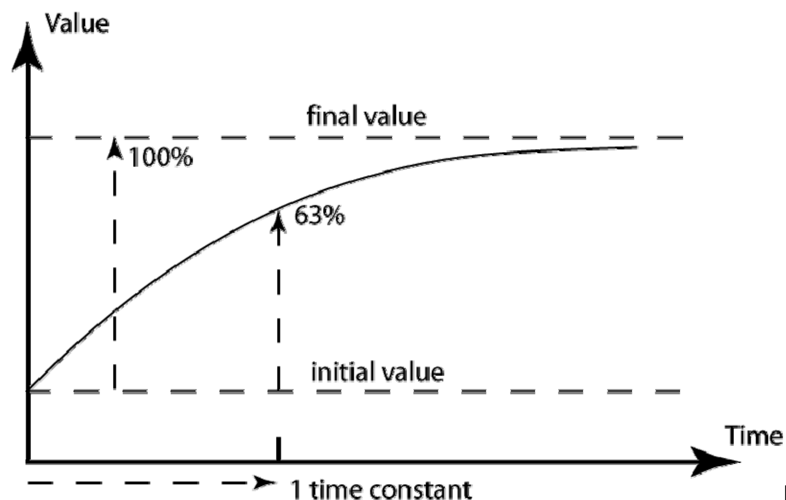

Fig 2

In a discrete time situation a low pass filter can be implemented by successive implementations of:

$$current\ output = \frac{(N-1)\,previous\ output}{N} + \frac{current\ input}{N} \qquad \text{equ. 4}$$

Where N is greater than 1, and N X (the iteration period) approximates to the time constant of the filter.

Equ 4 is of the same form as Dickinson's damping equation, [1, p27, equ. 5] – a proportional sharing of the current value and the input value.

$$new\ effective\ gas\ content = \frac{Z(ideal\ newcalculatedgascontent) + old\ gas\ content}{Z+1} \quad \text{equ. 5}$$

The relationship between N and Z is:

$$N = \frac{1}{Z} + 1 \quad \text{equ. 6}$$

Conveniently the iteration period in Ms Resi is 1 second, so N is the time constant of the filter in seconds.

In the Ms Resi implementation of the function,

```
float damp(float current_input, float previous_value, float time_constant),
```

the value of the time constant is displaced by one (this simplifies conversion from the Appendix IV code) such that the minimum valid value for the time constant is zero, equating to no additional delay besides the intrinsic delay of a finite sample period.