



Vigil

An Easy to Use Desktop Debugger

ProAction

3QD
DEVELOPMENTS LTD

APDL

Vigil

Copyright © Alan Wrigley, ProAction & 3QD Developments Ltd 2026
All rights reserved.

No part of this product may be reproduced in whole or part by any means without written permission of the publisher. Unauthorised hiring, renting, lending, public performance or broadcasting of this product or its parts is prohibited.

While every care is taken, the publisher cannot be held responsible for any errors in this product, or for loss of data or consequential effects from the use of this package.

Software and Documentation supplied by
APDL
a part of
3QD Developments Ltd
Single User version downloaded from the free of charge software section of
www.riscos.com

Please note that any free of charge downloads from RISCOS.com are supplied without any technical support package as per the licence agreement.

This document last edited on 29/09/26



If you have found this application and this manual useful please consider making a donation to help cover the costs of cataloguing, documenting, updating and hosting the APDL applications. Scanning the QR code will take you to our donations page.

Introduction

Vigil is a multi-tasking monitor/debugger for use with programs written in any language. Its unique combination of features lets you view and alter a program's workspace and variables while it is actually running on the RISC OS Desktop, something which is very difficult to do under normal circumstances. This makes it an ideal tool for people who are writing programs to run in the Desktop environment, and its power and user-friendliness makes it equally appropriate both for experienced users developing software for the commercial market, and for those trying to get to grips with the Wimp for the first time.

Since Vigil is aimed at programmers, this manual will assume that you have a basic knowledge of programming techniques, and in particular it will assume that you are familiar with the basic principles of programming a multi-tasking application.

You will need RISC OS 3.10 or later to run Vigil, but programs written for earlier versions of RISC OS (e.g. RISC OS 2.00) can be debugged using Vigil.

Vigil's main features

- Display the entire contents of an application's workspace (its Wimp slot). An update facility ensures that the display is updated whenever you need it. The workspace can be displayed in byte, word or mnemonic format, the latter being useful for ARM code programs. A search facility lets you search for a specified byte, word or string, and in the case of mnemonic display, a mnemonic or a particular SWI. Specified sections of the workspace can be printed or saved to disc either as raw data or as a text file. The contents of the workspace can be altered if required.
- For Basic programs it can display the current contents of the variables used by the program. The values of integer variables can be altered while the program is running.
- Display the current contents of the ARM processor registers. This is particularly useful for ARM code programmers, but may also be useful for Basic programmers, who can use the facility to track a particular variable or variables as the program runs.
- Display any or all Wimp messages received by the task being debugged. The user may choose which messages to include in the display. This lets you see exactly which reason codes are being returned by Wimp_Poll and when.
- Display the co-ordinates of the pointer, relative to both the screen and the window under the pointer, allowing you to calculate window co-ordinates more easily by reference to the window itself.
- Will give information about any module installed in the computer, and can display the module's code in disassembled format, the contents of the module's workspace, and a list of all SWIs and star commands recognised by the module. The workspace and code displays can be searched, printed or saved to disc in the same way as the application workspace display.
- Disassemble any ARM code file dragged to its icon (e.g. module, utility, absolute code program etc.).
- Has a simple centisecond timer which enables you to time events and improve time-critical sections of your code.
- Many of Vigil's facilities are available by simply clicking on an icon in its status window. Other facilities can be accessed directly from within your programs by SWI calls to Vigil.

Before describing Vigil's facilities in detail you must be aware of one consequence of the way the program operates. A multi-tasking application only has control of the computer at certain times, specifically in between receiving a poll reason code and handing back control by calling Wimp_Poll again.

Vigil is designed to provide debugging and monitoring facilities while your program is running in its normal environment, so it follows from this that any information sent to Vigil by your task, or vice versa, can only be displayed during the next poll loop. Depending on how this information is relayed, the display may reflect the exact state of affairs at the moment it was sent, or it may reflect the state of affairs at the point at which Wimp_Poll is next called.

This will all be made clearer in the detailed description, but it is important at this stage to be aware that information obtained from your program will not always necessarily be valid by the time it is displayed by Vigil. Provided you read these instructions carefully, this should not cause any problems.

Getting started

Vigil is run in the usual way by double-clicking on its icon in a Filer window. This installs it on the icon bar.

Note that every copy of Vigil has a unique serial number, which can be seen in the 'Info' window.

Only one copy of Vigil can be run at any time, and if you try to run a second copy an error message will be displayed. Note that it is possible that if you have been running Vigil and a machine error has caused it to quit unexpectedly, you may get this error on trying to re-install it. This is because Vigil's module is still in memory even though the application has quit. The remedy for this is to press **F12** to get to the command line, then type **RMKill Vigil** followed by **Return**.

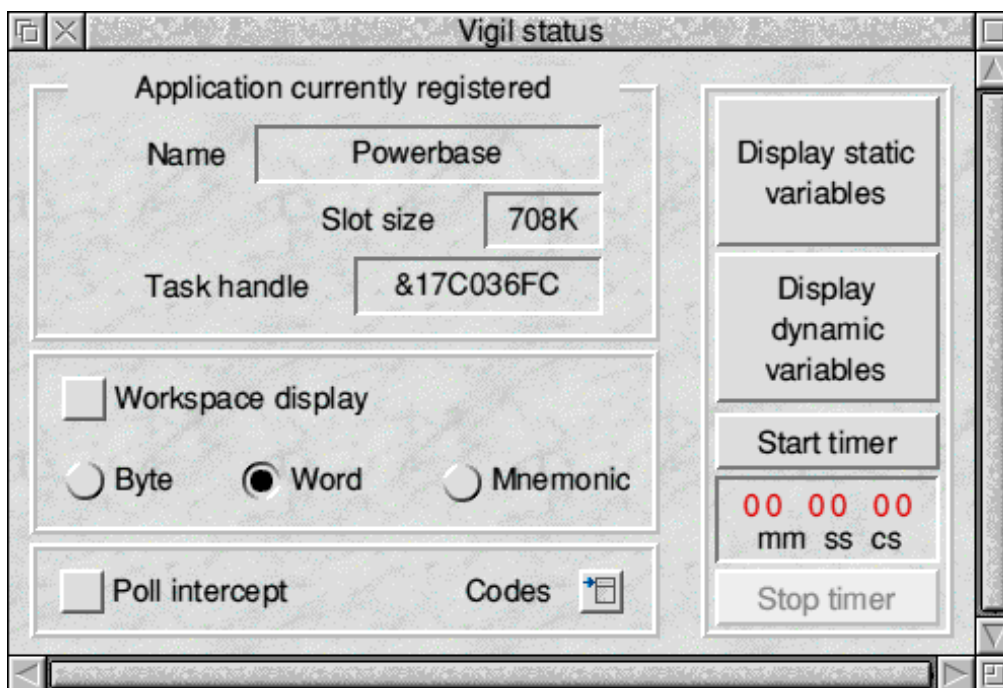
Once installed, clicking **Select** over the icon bar icon opens Vigil's Status window, which is used to display information about the task being debugged and to access certain facilities. However, the window will not contain any useful information until a task has been registered, as described below.

Clicking **Menu** over the icon bar icon displays a menu with five options. **Info** displays a dialogue box giving information about the program, including the version and serial number, which you should quote should you need to contact us. **Quit** terminates Vigil and removes its icon from the icon bar. The other three options, **Modules**, **Pointer** and **Save choices** will be described later, but first we will concentrate on using Vigil to debug a program.

Registering a task with Vigil

Before an application can make use of Vigil's facilities it must be registered with Vigil. This can be done by incorporating a SWI call, *Vigil_RegisterApplication*, in the program (see the section *SWI Calls*), or more simply by dragging the application directory icon belonging to the task to be registered onto Vigil's icon bar icon. Note that you must drag the application icon itself to Vigil, **not** an individual file such as the !RunImage. This will have a different effect, as described later in the section *Disassembling Files*.

If you register your task by calling the SWI *Vigil_RegisterApplication* this must be done after the task calls *Wimp_Initialise*. If you drag the application to Vigil's icon, it will be registered with Vigil and run at the same time. If the Status window is not already open, it will be opened when a task is registered, and the task's name, task handle and Wimp slot size will be shown.

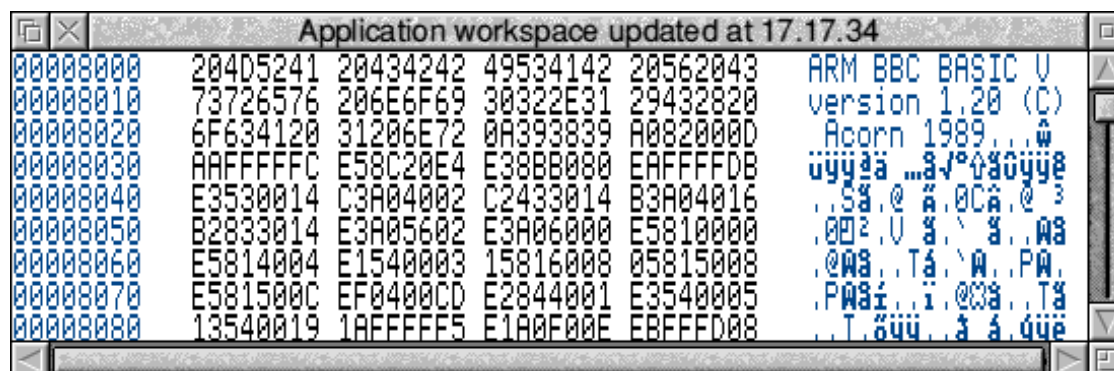


Once a task is registered, Vigil's debugging and monitoring facilities will be available to that task. Only one task can be registered at any one time. If a second task is registered, the first will automatically be de-registered.

Viewing application workspace

All Wimp applications are allocated a chunk of memory in which to work; this is known as the *Wimp Slot*. The program usually claims the amount of memory it needs to run in its !Run file. Provided enough memory is available the operating system allocates a block to the task. This may be anywhere in the computer's memory, but the OS performs a sophisticated form of mapping so the task always thinks its memory starts at &8000. So, irrespective of the actual physical address of the memory, every task running on the Desktop thinks its own memory starts at &8000 and extends in a continuous block the size of the Wimp slot.

Vigil lets you view the contents of the registered task's workspace but there must be enough free memory in the computer when the task is registered to enable Vigil to keep a copy of the entire workspace. So, if your program requires 32K of memory 64K must be available before Vigil can display the workspace contents. If there is not enough memory a warning is given and you will be offered the option of registering the task but without the workspace display facility.

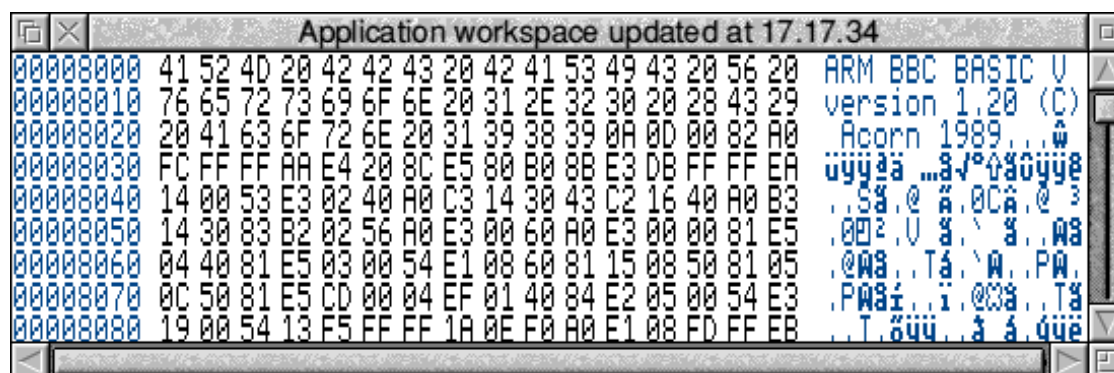


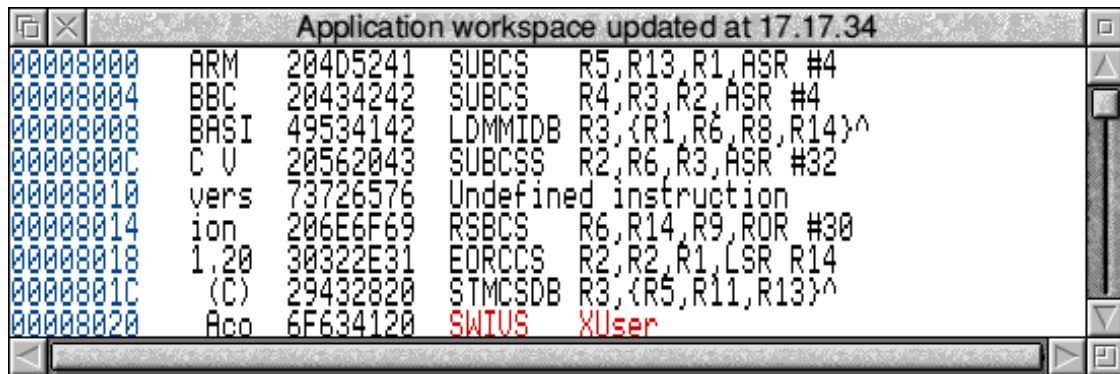
By default, Vigil starts up in word format (above) but you can alter this by changing format and then choosing the Save choices option from the icon bar menu. This saves the currently selected format as the default for future use.

If the caret is not currently visible in any other window pressing **Page Up** or **Page Down** will scroll the window up or down by one screen.

Assuming that there is enough memory you can open the workspace display window by switching on the Workspace display icon in the Status window. You can also choose from the Status window whether the display is to be in byte, word or mnemonic format. Byte format display shows the hex contents of each byte separately in rows of 16, together with the ASCII character which corresponds to each byte's value if it is a printable character. Word format is similar, but in this case the hex display is shown word by word (i.e. four bytes at a time). This form of display can be very useful since most processes in RISC OS work on four-byte (i.e. 32-bit) values. However, if you are not used to this form of display you should use it with care, since a four-byte hex value is always shown with its most significant byte first, as you would expect, whereas the ASCII character display for that word will be the other way round, since the word is actually stored in memory with the least significant byte first.

Mnemonic format assumes that the contents of the workspace are a sequence of ARM code instructions, and displays them as a disassembled listing. This is useful if the registered task has been written in ARM code, or has been compiled into ARM code from a higher-level language.





There are many reasons why you might find the application workspace display useful when developing programs. Often you will need to know exactly what is going on inside your program's memory while it is running. For example, you may be having a problem with a Wimp SWI call, and want to see just what values are being passed in the parameter block. Having found out the address of the start of the block you can then set up the workspace display window to show that part of memory, and see what happens to the contents as you perform various actions.

While the program is running, of course, you would not normally know the exact address in memory at which the block is located. With a Basic program you can find this out quite simply by first using the variable display function, (described in the section Basic Variable Display). For example, if the parameter block is situated at block%, just use the variable display to find out the value of block% and then go to that address in the workspace display. For ARM code programs, load a register with the address of the block and then use the register display facility (see the section ARM Register Display), for example:

```
ADR R0,block:SWI "Vigil_DisplayRegisters"
The address of the block will now be shown in R0.
```

To take another example, you may have an ARM code program which crashes at a certain address with an "Address exception" error. By viewing, in mnemonic format, the instruction at the relevant address you may be able to see exactly what is going wrong. A word of warning is necessary here - the address given in the error message is not actually the offending address. This is because of the pipelining effect of the ARM processor, by which the instruction being executed is two steps behind the program counter. For this reason, if you see an error message such as "Address exception at &90D8", then you must look at &90D0 for the instruction which actually caused the error.

Clicking **Menu** over the application workspace window opens a menu with ten options. Most of these are duplicated by a toolbox attached to the left-hand side of the window. There are five tools arranged down the left-hand side of the toolbox; from top to bottom, these are the Format tool, the Goto start tool, the Goto end tool, the Goto tool and the Update tool. On the right-hand side there are six tools. From top to bottom, the Search next tool, the Search previous tool, the Clear tool, the Write tool, the Print tool and the Save tool. The functions of these will be described with the menu options to which they relate.

The Main Menu

The menu options are as follows:

Format

This leads to a dialogue box where you can choose between byte, word or mnemonic display formats. It duplicates the functions of the icons in the Status window. Clicking with **Adjust** on the Format tool also opens the dialogue box. Clicking on Format on the menu without displaying the dialogue box or clicking with **Select** on the Format tool cycles through the three display formats.

Go to start

Choosing this or clicking on the Goto start tool forces the display to the start of the application workspace, i.e. address &8000. If the caret is not visible in another window pressing **Ctrl + Up Arrow** has the same effect.

Go to end

Choosing this or clicking on the Goto end tool forces the display to the end of the application workspace. The address of this will depend on the size of the task's Wimp slot. For example, with a Wimp slot of 32K the last address shown will be &FFFF (&8000+32K=&10000). If the caret is not currently visible in any other window pressing **Ctrl + Down Arrow** has the same effect.

Go to

This leads to a dialogue box allowing you to specify a particular address within the application memory. Clicking with **Adjust** on the Goto tool also opens the dialogue box. Pressing **Return** in the box, or clicking on its **OK** icon, forces the display to jump to that address. Clicking on the Go to menu option directly or clicking with **Select** on the Goto tool, forces the display to the last specified address in the dialogue box, unless that address is not valid, in which case the dialogue box will be opened.

Update

This leads to a dialogue box where you can update the workspace display. This means that the current contents of the task's workspace are transferred to Vigil's store and the window display is updated to reflect the new contents. The window title bar gives the last update time to the nearest second. Clicking on the Update tool with **Adjust** also opens the dialogue box. The dialogue box offers a choice between single update, which means the contents are updated once when you ask for it to be done, or periodic update, which updates the contents regularly at specified intervals. Clicking on Single update selects the former, while clicking on Update every selects the latter and lets you enter a value. This should be a whole number and will be the number of seconds between each update.

Clicking on the Update icon will immediately update the display once if single update is selected, or will start the periodic update process if periodic update is selected. Clicking on Freeze will stop periodic update if it is currently in operation.

Clicking on the Update menu option or clicking with **Select** on the Update tool, will update the display once if Single update is selected, or will toggle the update on or off if Periodic update is selected. In other words, if update is off it will duplicate the Update icon, while if update is on it will duplicate the Freeze icon.

Search

This leads to a dialogue box which lets you search through the application workspace for a byte, word (4-byte value) or text string and, if mnemonic display format is in force, for a particular mnemonic or SWI call. The dialogue box is shown below. Clicking with **Adjust** on either the Search next or Search previous tools also opens the dialogue box.

On the left-hand side is a group of radio icons which let you choose what type of object you want to search for. If byte or word format display is in force the Mnemonic and SWI search options will not be selectable. When one of the options is selected the caret appears in the writable icon to the right of that option's icon, allowing you to type in the value or text you are searching for. Note that for byte and word searches the value must be in hex, while for mnemonic and string searches the case of the characters is not significant - "test" will also find "Test", "TEST" etc.

However, when searching for a SWI, the case must be exact, just as when you are using SWI calls in your programs. The string search finds any occurrence of the given sequence of characters, even if they are part of a longer string. The mnemonic search matches from the start of a disassembled line and is wildcarded to the end

of the line - i.e. "B" will find BL, BNE, BIC etc., while "BIC R0" will only find BIC R0...

To begin a search click on one of the icons at the bottom of the window. Clicking on Next or clicking with **Select** on the Search next tool finds the next occurrence of the object starting from the address currently displayed at the top of the application workspace window. Clicking on Previous, or clicking with **Select** on the Search previous tool, finds the previous occurrence of the object starting from the same address. Clicking on Search from start searches from the beginning of the application workspace (i.e. &8000). When a search object is found, the memory which contains it becomes the currently marked section.

Clear

Choosing this or clicking on the Clear tool clears any currently selected word or block of memory, and redraws the window if the marked section is visible.

Write

This leads to a dialogue box where you can alter the contents of memory in units of one word (4 bytes). This can also be opened by clicking on the Write tool with **Adjust**. Enter the start address of the write operation opposite Start at, and the end address opposite End at. Note that the end address is exclusive, i.e. Vigil will write up to but not including the specified address. Enter the required value and click on OK or Cancel to abort the operation. If OK is selected the specified value will be written to each word in memory between the start and end addresses.

Clicking directly on the Write menu option or clicking on the Write tool with **Select** will write the last values given directly to memory using the last values given in the box, though a warning will be given as this could be destructive. If the values are not valid, the dialogue box will be opened instead.

The Write facility is useful if you want to initialise a block of memory to zero or another value. You could also use it to fill a parameter block with different values without having to quit the program. For example, you could have a window definition stored in a parameter block and could adjust it before opening the window to enable you to try out various effects while the program is running.

Print

This opens a dialogue box which lets you print out any part of application workspace in the same format as currently displayed in the window. Clicking on the Print tool with **Adjust** also opens the dialogue box. Icons at the top let you choose between *All*, when the whole workspace is printed, and *Selection*, which lets you to enter start and end addresses. Clicking on the Print icon will print the information as a text file via the RISC OS printer manager if it is installed; if not it will print directly to the currently selected printer port. This means you do not need to have the printer manager running to print from Vigil. Clicking on Cancel will abort the operation. Clicking directly on the Print menu option, or clicking on the Print tool in the toolbox, duplicates the effect of the dialogue box's Print icon.

Save

This leads to a dialogue box where you can save all or part of the application workspace, either as a text file in the same format as the current display format or as raw data. Clicking with **Adjust** on the Save tool also opens the dialogue box. The upper part of the box is identical to, and performs the same functions as, the Print dialogue box just described. The central part contains two radio icons which let you choose between text or raw data format. To save either drag the file icon to a Filer window, or press **Return** or click on OK if the writable icon contains a full pathname. Clicking directly on the Save menu option, or clicking on the Save tool, activates the save automatically.

The other menu options work as described for the application workspace window. There is a toolbox attached to the window duplicating the functions of the menu items, again as described earlier. Note that where you must specify an address, for example in the Go to, Print or Save dialogue boxes, the address must be valid for the current display base; in other words, if the base is currently zero, it must lie between zero and the last address shown in the window's work area, while if the base is not zero the address must be greater than or equal to the lowest displayed address and less than or equal to the highest.

Application workspace display functions

The effect of clicking **Select** over the application workspace window depends on which display format is in force. For Byte format a caret is placed in the window to let you alter the contents of the workspace by typing characters or values directly into the window. This is described below in the section *Altering Single Bytes*. For word and mnemonic format, using **Select** or **Adjust** over the window lets you mark a word or section of memory which can be written to using one of the functions described below.

In Mnemonic format, clicking with **Select** anywhere in the window selects and marks the word of memory displayed on the same line as the pointer (a marked section is shown in reversed colours). You can also mark a block of memory by dragging the pointer down the window with **Select** or **Adjust**, or by marking the start of the block with **Select** and the end with **Adjust**.

In Word format, clicking within the left- and right-hand sections of the window (where the text is blue) selects and marks the whole line (i.e. 4 words), while clicking in the central area selects just the word under the pointer. As with the mnemonic format, a block can be marked by dragging or by a combination of **Select** and **Adjust**. When a word or block is marked, a dialogue box pops up to allow you to alter the memory contents. This is described later.

Altering single bytes

Although writing directly to a word or block of memory as described above is very useful there are times when you may want to alter just a single byte, or to type ASCII characters directly into memory. This can be done using the byte display.

Clicking **Select** over the window in byte format places a caret before the character under the pointer, either in the hex or the ASCII section. You may now type directly into the workspace. If the caret is in the ASCII section you can enter any character from the keyboard, including control characters by using **Alt** and the numeric keypad. The character entered will replace the existing character but will be shown as a dot if it is unprintable, i.e. below ASCII 32. The caret will move to the next byte, and the hex section will be updated to reflect the new character. The **Delete** key will delete the character to the left, just as you would expect.

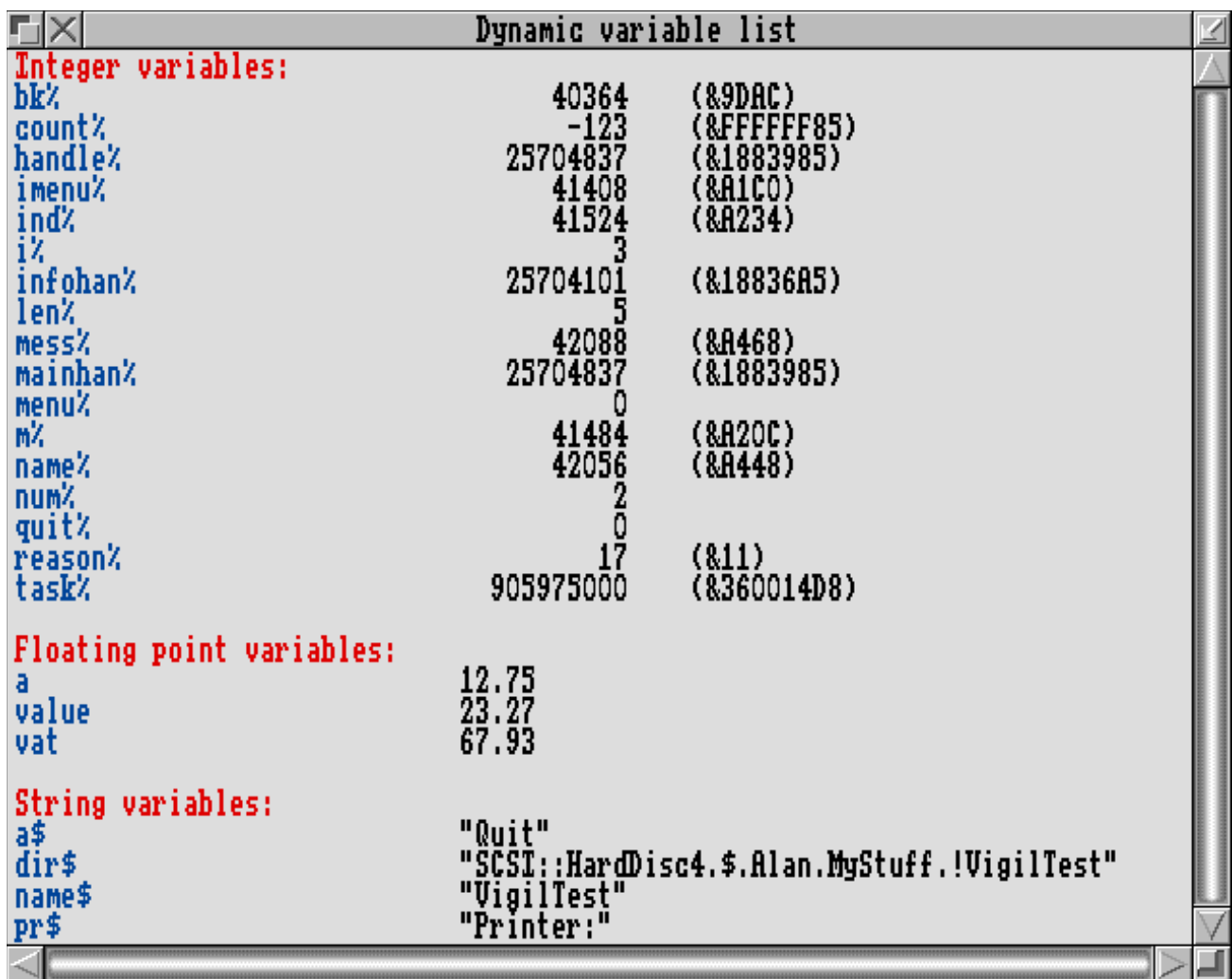
If the caret is in the hex section only valid hex characters can be entered. Each digit can be altered separately. Pressing **Tab** will switch the caret from hex to ASCII or vice versa at the same address. You can move around the visible area of the window by using the cursor keys.

Each time you type a character into the window the task's application memory will be updated immediately.

BASIC variable display

If the registered task is a Basic program you can view the current contents of any variables known to the program. This can be done in one of two ways, either by clicking on the appropriate icon in Vigil's Status window, or by calling the appropriate SWI from within your own program (see the section SWI Calls). The advantage of using the latter method is that you can see the values of the variables at a precise point in the program (remembering that Vigil can only interrogate your program when it next relinquishes control by calling Wimp_Poll, and the values may have altered in the meantime).

Two options are available, static variable display and dynamic variable display. Static display shows the values of the resident integer variables, @% and A%-Z%, while dynamic display shows all other integer, floating point (to an accuracy of 8 digits) and string variables. Arrays are not shown. Note that for dynamic variable display only those variables which are currently known to the program will be shown - if a variable is first declared in a section of the program that has not yet been executed, Basic will not know of its existence, and it will not have a value or appear in the display. Also, the order of the variable names is the order in which they are stored in the Basic workspace, and will therefore only be sorted on the first character. This means that although all variables starting with B, for example, will appear after those starting with A, you might find that menu% appears before main%. A typical dynamic variable display is shown below.



Clicking with **Select** over any variable in the static display window, or over an integer variable in the dynamic display window, opens a dialogue box enabling you to alter the value of that variable. The value is updated immediately. This facility lets you try out various values as parameters within your program, or to correct values that are not as expected.

The facility to select an individual variable or set of variables for display is not currently implemented, though this is scheduled for a future version. In the meantime, however, it is very easy to achieve by using the ARM register display facility, described next.

ARM register display

By using the SWI call *Vigil_DisplayRegisters* (see the section SWI Calls), you can view the contents of the ARM processor's user-mode registers at any instant. Note that, unlike most of Vigil's SWI calls, this SWI may be called by any task whether it is registered with Vigil or not. The values displayed are those at the actual time of making the call, irrespective of when Vigil next gains control of the computer. This facility is extremely useful for ARM code programmers, since it is often very frustrating knowing that something is not working correctly but being largely powerless to see what is happening while the program is running. If you are developing ARM code programs, you will probably find this facility one of Vigil's most useful features. By inserting a single SWI call at any point in the program, you can see precisely what all the registers are holding at that particular point.

However, Basic programmers may also find this facility useful. When you make a SYS call in Basic, you can specify the entry contents of registers R0-R7. This means that you can use the call to display the current contents of up to 8 different variables by giving them as parameters. For example:

```
SYS "Vigil_DisplayRegisters",a%,block%,out%,x%
```

would show the value of a% in R0, block% in R1, out% in R2 and x% in R3. By making the call just before or after each call to Wimp_Poll, you can keep track of a set of variables throughout the execution of your program.

You can also compare the values obtained by the register display with the values obtained at another point in the program, by using the SWI *Vigil_CompareRegisters*. This SWI compares the register contents at the point of calling with the values obtained at the last call to *Vigil_DisplayRegisters*, and highlights any differences in red. If there has been no previous call to *Vigil_DisplayRegisters*, the error message *"No previous register values to compare"* is given.

Wimp message display

A useful feature of Vigil is the ability to see exactly what messages your task is receiving from the Wimp, and when. This can be activated from the Status window by switching on the Poll filter icon (and also by a SWI call from your program - see the section SWI Calls). Vigil will now intercept poll reason codes sent to your task and display information about them in a window. By default all poll codes are intercepted, but you may choose which you want to intercept by clicking over the arrow icon labelled Codes. This opens a window displaying all the poll codes currently used by RISC OS, together with an on/off icon for each. If the icon is switched on, then the corresponding reason code is intercepted. Clicking on the OK icon updates your choice of codes, while clicking on Cancel restores the icons to the original settings. You can alter the start-up default set by choosing the Save choices option from the icon bar menu; this will save the current set of reason codes which will become the default in future.

The poll code display window is shown in Figure 6. As messages are received from the Wimp by the registered task they are logged in the window, starting at the top. The window can show up to 16 messages. When it is full new messages will appear at the bottom and older ones pushed from the top. The information displayed is the time at which the message was received to the nearest centisecond, the poll reason code, the textual description of the reason code, and in the case of user messages, the message action code to enable you to identify it.



Time	Reason code	Message	Action code
10.56.33.10	2	Open_Window_Request	
10.56.36.39	6	Mouse_Click	
10.56.40.19	8	Key_Pressed	
10.56.50.50	17	User_Message	&400C1
10.57.01.34	6	Mouse_Click	
10.57.03.48	9	Menu_Selection	

As Vigil can only display these messages when it next gains control through Wimp_Poll, and as Wimp messages are given priority over other reason codes when switching between tasks, there may be times when the registered task has received more than one message before Vigil gets the chance to display them. Vigil can cope with up to 5 consecutive messages in this way, which should cater for almost all situations, but if more than 5 occur in one sequence then the earlier ones will not be displayed though they will still be sent to the registered task.

If you make any alterations to the poll mask while the Wimp messages window is open, these will take effect immediately. So if you switch off, say, Mouse_Click (reason code 6), then mouse clicks will no longer be shown on the messages display. Closing the messages window cancels the intercept process.

Other functions

Timer

The other feature accessible from the Status window is the centisecond timer. This can be switched on by clicking on the Start timer icon, and the elapsed time will be shown in the digital display below it. Please note that if you perform any tasks on the Desktop that take a while to complete, or if you leave the Desktop, the timer display will temporarily stop updating. However, the timer will continue to count, and the display will be updated to the correct elapsed time as soon as it is possible to do so. The timer can be stopped by clicking on the Stop timer icon. Both these functions can also be carried out from within the registered task by using SWI calls. This is very useful if you want to know exactly how long a particular process will take, as you can call *Vigil_StartTimer* at the start of the process and *Vigil_StopTimer* at the end. Note that, unlike most of Vigil's SWI calls, these SWIs may be called by any task whether it is registered with Vigil or not.

Pointer co-ordinates

Choosing the Pointer option from the icon bar menu opens a window which displays the current co-ordinates of the pointer in OS units. Three sets of co-ordinates are shown: Screen, Relative and Work area. Screen co-ordinates are relative to the screen origin (the bottom left-hand corner). Relative co-ordinates are relative to a user-defined point on the screen. Initially this is the same as the screen origin, but by pressing **Alt-Home** you can set the Relative origin to the current position of the pointer. Thus you can move the pointer to a particular point on the screen, press **Alt-Home**, and see the co-ordinates relative to that point. The Work area co-ordinates are relative to the work area origin of the window currently under the pointer. A further pair of values in the Pointer window gives the window and icon handles currently under the pointer. All the values in the window can be displayed in either decimal or hex, and a pair of radio icons at the bottom of the window allows you to choose between the two.

Module information

Choosing the Modules option from the icon bar menu opens a window listing the names of all modules currently installed in the computer. Clicking with **Select** on any module name in the list opens a Module information window, giving details of the module, including its version number and the address and length of its code and workspace areas. If the module has more than one instantiation, an arrow icon allows you to cycle through all the instantiations of that module.

Four action icons at the bottom of the Module information window allow you to view the contents of the module's code (in disassembled format), and of its workspace, and also list all the SWI calls and star commands recognised by that module. If any of these options is not available, e.g. it has no workspace or recognises no SWIs, the action icon will be greyed out and will not be selectable. Only one of these options can be displayed at any one time, so if you select a second action icon the window opened by the first will be closed.

The Module information window also incorporates a standard RISC OS save box which enables you to save the module's code as a file.

Clicking **Menu** over the module code and module workspace display windows opens a menu which is similar to the menu for the application workspace window described earlier. However, the Write and Clear options are missing, and in the module code window, the Update option is not available as the module's code is fixed. In place of the Format option is a new option entitled Base zero. This enables you to switch the displayed start address of the memory contents between zero and the actual address. Using the actual address allows you to see exactly where in memory it is situated, while using the zero base option makes it easier to calculate offsets into the module, for example, or to specify sections to be printed or saved. The menu item is ticked when zero base display is selected.

Disassembling a file

If you drag a file rather than an application icon to Vigil's icon Vigil will assume that the file is to be displayed in disassembled format. If it is a known ARM code file type (Module, Utility or Absolute) it will be displayed as a disassembled listing in a window. If it is any other file type, a warning will be given, together with an option to disassemble it anyway. Module and Utility files will be assumed to start at zero and the disassembled listing will start with that address. Absolute files will be assumed to be applications and disassembled with a start address of &8000.

The disassembly window has a menu structure and a toolbox identical to the one described above for the module code window. In this case, choosing the Base zero option toggles between zero and &8000.

SWI calls

Vigil supports a number of SWI calls which make it easy to request debugging information directly from within your program. These are described overleaf. Many of the calls have no entry parameters; where this is not the case, the entry parameters are shown. In all cases, the register contents are preserved on exit.

- **SWI Vigil_RegisterApplication** **&46100**
On entry: R0=task handle
This call is used to register an application with Vigil.
- **SWI Vigil_DeregisterApplication** **&46101**
This call may only be used by a task registered with Vigil. It is used to de-register the currently-registered application.
- **SWI Vigil_DisplayRegisters** **&46102**
This call displays the contents of the user-mode ARM registers.
- **SWI Vigil_CompareRegisters** **&46103**
This call compares the register contents with the last values obtained from Vigil_DisplayRegisters, and highlights any differences in red. If no previous call has been made to Vigil_DisplayRegisters, an error is given.
- **SWI Vigil_DisplayMemory** **&46104**
On entry: R0 = start address
This may only be used by a task registered with Vigil. It is used to display a block of 256 bytes of memory starting at the address given in R0.
- **SWI Vigil_CompareMemory** **&46105**
This call may only be used by a task registered with Vigil. It compares the contents of the block last displayed by Vigil_DisplayMemory, and highlights any differences in red. If no previous call has been made to Vigil_DisplayMemory, an error is given.
- **SWI Vigil_StaticVariableList** **&46106**
This call may only be used by a task registered with Vigil. It displays the current contents of the static variables @%-Z%. It can only be used by Basic programs, and if the registered task is not Basic, an error is given.
- **SWI Vigil_DynamicVariableList** **&46107**
This call may only be used by a task registered with Vigil. It displays current contents of all dynamic variables currently known to the program. It can only be used by Basic programs, and if the registered task is not Basic, an error is given.
- **SWI Vigil_StartTimer** **&46108**
This call is used to start the interval timer.
- **SWI Vigil_StopTimer** **&46109**
This call is used to stop the interval timer.
- **SWI Vigil_Delay** **&4610A**
On entry: R0 = number of centiseconds to pause
This call pauses operation of the computer for the specified time.
- **SWI Vigil_FilterOn** **&4610B**
On entry: R0 = poll filter mask
This call sets up a post-filter on the registered task to intercept and display Wimp messages received by the task, as described in section 9, Wimp Message Display. For more details on poll codes and the poll code mask, see the Programmer's Reference Manual.
- **SWI Vigil_FilterOff** **&4610C**
This call may only be called by a task registered with Vigil. It cancels the post-filter and turns off the display of Wimp messages.

All the SWI calls recognised by Vigil act immediately - the contents of registers, memory, variables etc. are those at the actual time of making the call. However, Vigil can only handle one call per poll loop, since the results of that call have to be displayed next time Vigil gains control of the computer. So if you make, say, a call to Vigil_DisplayRegisters and a call to Vigil_StartTimer during the same poll loop, only the second call will be acted upon.

The exception to this is if the second call is a comparison (Vigil_CompareRegisters or Vigil_CompareMemory), and the first call was the appropriate display call with which the comparison is to be made (i.e. Vigil_DisplayRegisters or Vigil_DisplayMemory). This allows you to compare two conditions within a short space of each other, for example the contents of the registers before and after calling a subroutine or making a SWI call.

Here are just a few of the other products you will be able to
download from RISCOS.com free of charge
(You can make a donation to help support future product releases but this is entirely voluntary)



This Documentation is © 3QD Developments Ltd 2026
Some of the products shown above might not be available immediately.
Not all products will run on all versions of RISC OS, check the product information page for details.

Other Products list last updated 29/2/26