DevHealth Memory Usage Tool for Windows Embedded Compact 7

Writer: Windows Embedded Compact JDP Team

Published: February 2012

Applies To: Windows Embedded Compact 7

Abstract

This paper describes how to use the DevHealth tool to analyze memory use on your Windows Embedded Compact device. It includes:

- An overview of the DevHealth tool and the types of reports that it generates.
- Guidance on how to run the tool.
- Descriptions of the individual reports.

Contents

Introduction 3

Using DevHealth 4

Running DevHealth 4

Interpreting the Report Data 6
  System Memory Report 6
  System Memory Map Report 9
  Process Report 12
  Module Report 13
  Heap Report 15
  Dependency Report 17

Conclusion 18

Additional Resources 19

Introduction

DevHealth is a memory-reporting tool that you can run on Windows Embedded Compact 7 devices with or without establishing a connection between your development computer and your device. By using DevHealth, you can take a snapshot of virtual memory and generate an overview of the memory footprint of the device. DevHealth can produce the following reports, which are text files that you can read by using any text file viewer:

- **System Memory Report:** Lists the amount of memory on the device and how it is being used
- **System Memory Map Report:** Shows the page types of the virtual memory that is being used by each process
- **Process Report:** Lists the number of pages of virtual memory that are being used by each process
- **Module Report:** Lists the number of pages of virtual memory that are being used by each DLL module
- **Heap Report:** Provides information on each heap that is being used by each process
- **Dependency Report:** Lists the module reference counts for all processes and DLL modules that are loaded into memory

You can use DevHealth to monitor the memory usage of the device or as the starting point for investigations into memory leaks. For example, to monitor device or application memory usage over a long period of time, you can run DevHealth at 15-minute intervals by using the timer option. You can then analyze the individual output files. You can also use DevHealth for memory leak investigations by comparing DevHealth reports before and after a complex scenario. An unexpected increase in the program memory usage may indicate potential memory leaks. Note that DevHealth reports memory usage on a module-by-module and process-by-process basis and does not provide more specific details about individual memory allocations. However, this information may help you narrow down memory leaks to processes or modules. At that point, you can use other tools such as the Resource Leak Detector in the Windows Embedded Compact Remote Tools Framework to find exactly where the memory leaks are located in the source code.

Other Windows Embedded Compact 7 tools may generate data similar to what DevHealth generates. For example, you can use the Performance Monitor or the Resource Consumer in the Remote Tools Framework to monitor system memory usage. The Target Control command `mi` in Platform Builder also generates memory map data. However, those methods require a connection between your development computer and your device, and they report less detailed memory usage information than DevHealth. DevHealth is not designed to replace these other tools, but to provide an additional option. You should select your tool based on your needs.

Using DevHealth

DevHealth consists of two device-side files: the main program (devhealth.exe) and a kernel DLL (devhealthdll.dll). You can find these files on your development computer in %_WINCE700%\public\COMMON\oak\target\<CPU Type>\<Build Type>, where <Build Type> is checked, debug, or retail.

You can run DevHealth either from your device or from Platform Builder. If you want to run DevHealth from your device, you need to copy both devhealth.exe and devhealthdll.dll to the device. Because devhealthdll.dll is a kernel DLL, copy it to the Windows directory on the device. If you plan to run the tool from the device with command-line options, and you do not have the Standard Shell (SYSGEN_STANDARDSHELL) in your OS image, include the Command Processor (SYSGEN_CMD) in your OS image. If you want to run DevHealth from Platform Builder by using a kernel independent transport layer (KITL) connection, copy devhealth.exe and devhealthdll.dll to the flat release directory.

Running DevHealth

You can run DevHealth with command-line options to produce different memory usage reports. When you run DevHealth with no command-line options, the output is the same as when you use the option `all`. The command-line options are listed in the table below.

Command-line option	Result	Report description (if applicable)
all	Produces all of the reports listed in this table. The output is the same as if there were no command-line options.	Each report is described in this table.
system	Produces a System Memory Report	This report lists the amount of RAM and non-volatile storage that is on the device and provides an overview of the number of pages that are used by each consumer of RAM.
map	Produces a System Memory Map Report	This report shows the individual page types of the virtual memory that is used by each process.
proc	Produces a Process Report	For each process, this report lists the number of pages of virtual memory that are used for

Command-line option	Result	Report description (if applicable)
		a subset of page types.
mod	Produces a Module Report	For each DLL module, this report lists the number of pages of virtual memory that are used for a subset of page types.
heap	Produces a Heap Report	For each heap that is used by each process, this report lists the starting and ending address and the number of blocks that are allocated for each module.
depend	Produces a Dependency Report	This report lists the module reference counts for all processes and DLL modules that are loaded into memory.
ignoredup	Ignores duplicate pages	None.
pte	Displays page table entries	None.
timer	Generates a report every 15 minutes	None.

The procedures below show you how to run DevHealth on your device or by using Platform Builder on your development computer. If you use Platform Builder, make sure that your development computer is connected to your device before starting that procedure.

To run DevHealth on your device

1. On your device, open a Command Prompt window.
2. At the command prompt, type `devhealth.exe <command-line options>`

To run DevHealth from Platform Builder

1. In Platform Builder, click **Target** and then click **Target Control**.
2. At the Target Control command prompt, type `s devhealth.exe <command-line options>`

Interpreting the Report Data

DevHealth reports are text files that you can view by using any text file viewer. Every time you run DevHealth, it creates an output text file named mem_*.txt, where the wildcard is a number based on how many mem_*.txt files already exist in the directory.

Note

Every time you run DevHealth, it creates just one output file (mem_*.txt), which contains all of the individual reports that you chose to run based on your command-line options.

DevHealth puts its output file in one of the following locations on the device, whichever one it finds first:

1. \Storage Card
2. \Hard Disk
3. \Release

   Note

   If you run DevHealth while connected to Platform Builder over a KITL transport, you can access your device's \Release directory on your development computer by going to the flat release directory, which is the same as the \Release directory on your device. The flat release directory is a single directory on your development computer that contains all of the files to be included in the final OS image, specified as the **Release directory** in the **<My Project>** **Property Pages** dialog box in Platform Builder.

4. Device root directory

For a description of the data that each report generates and how to interpret that data, see:

- System Memory Report
- System Memory Map Report
- Process Report
- Module Report
- Heap Report
- Dependency Report

System Memory Report

The DevHealth System Memory Report provides a summary of the amount of RAM and non-volatile storage on the device and shows which consumers are using the physical RAM at the time that you run the DevHealth tool. There are three sections in the System Memory Report: a total storage section and two physical RAM sections, as described below.

- **Total Storage:** Provides the total number of pages of non-volatile storage space and the number of available and used pages of non-volatile storage space.

- **Physical RAM (sources breakdown):** Lists the sources of RAM that are on the device. This section always includes a Main Memory section, and may include one or more Extension DRAM sections.
- **Physical RAM (consumers breakdown):** Shows how the entire physical RAM of the device is being used at the moment of the report. This part of the report breaks the memory down into different sections such as program memory, object store, paging pool, Watson size, and so on. Within the Program Memory section, there is a summary for each type of page that is allocated, such as code, stack, heap, data, and so on.

The address ranges for the various sections of RAM correspond to the actual physical addresses that each virtual address range maps to. In this way, this report can account for the entire physical RAM on the device at any time.

In the System Memory Report, "Unaccounted/Unknown" indicates that there is a difference between the number of physical pages that are accounted for and the number of physical pages that are available. That is, there are some pages for which the usage is unknown. This is different from "unknown" in the System Memory Map report. In the System Memory Map Report, "unknown" means that DevHealth found pages in use but cannot report their purpose. In the System Memory Report, "unknown" means that there are some pages that DevHealth did not even find.

One possible source of memory listed as "Unaccounted/Unknown" is by processes that have exited. If a process exits, but a program still has a handle to that process, the memory of the process is not freed, but the process will not show up in DevHealth report. As a result, a large amount of unknown memory could indicate a leak of process handles.

An example of a System Memory Report is shown below.

```
Page Size:   4096
                                     |   Pages |  Size (bytes) |  Size (MB) |              Address
 Total Storage (Flash)               |   28838 |     118120448 |     112.65 |                  n/a
   Available Storage                 |   28820 |     118046720 |     112.58 |                  n/a
   Used Storage                      |      18 |         73728 |       0.07 |                  n/a

 Physical RAM (sources breakdown)    |   57839 |     236908544 |     225.93 |                  n/a
   Main Memory                       |    8687 |      35581952 |      33.93 | 0x81d00000 - 0x83eeefff
   Extension DRAM 1                  |   49152 |     201326592 |     192.00 | 0x94000000 - 0x9fffffff

 Physical RAM (consumers breakdown)  |   57839 |     236908544 |     225.93 |                  n/a
   Kernel Prealloc.                  |      53 |        217088 |       0.21 | 0x81d00000 - 0x81d34fff
   Page Tables                       |       1 |          4096 |       0.00 | 0x81d35000 - 0x81d35fff
   Kernel Log Ptr                    |       1 |          4096 |       0.00 | 0x81d36000 - 0x81d36fff
   Watson Size                       |       0 |             0 |       0.00 |                  n/a
   Overhead (kernel RAM map)         |      29 |        118784 |       0.11 |                  n/a
   Object Store                      |   28877 |     118280192 |     112.80 |                  n/a
   Program Memory                    |   28878 |     118284288 |     112.80 |                  n/a
     AVAILABLE PROGRAM MEMORY        |   26441 |     108302336 |     103.29 |                  n/a
       'Free' pages in-use by pool   |       0 |             0 |       0.00 |                  n/a
     USED PROGRAM MEMORY             |    2437 |       9981952 |       9.52 |                  n/a
       Kernel Objects                |      91 |        372736 |       0.36 |                  n/a
       Unused Paging Pool            |     794 |       3252224 |       3.10 |                  n/a
       Unaccounted / Unknown         |     742 |       3039232 |       2.90 |                  n/a
       Programs                      |    1604 |       6569984 |       6.27 |                  n/a
            (S) Stack                |     169 |        692224 |       0.66 |                  n/a
            (H) Heap                 |     538 |       2203648 |       2.10 |                  n/a
            (E) EXE Data             |       6 |         24576 |       0.02 |                  n/a
            (D) DLL Data             |     379 |       1552384 |       1.48 |                  n/a
            (c) Code RAM             |       2 |          8192 |       0.01 |                  n/a
              Process                |       2 |          8192 |       0.01 |                  n/a
              Module                 |       0 |             0 |       0.00 |                  n/a
            (r) Read only RAM        |     175 |        716800 |       0.68 |                  n/a
              Process                |       7 |         28672 |       0.03 |                  n/a
              Module                 |     168 |        688128 |       0.66 |                  n/a
              Map/Shared             |       0 |             0 |       0.00 |                  n/a
            (W) Read/Write RAM       |     335 |       1372160 |       1.31 |                  n/a
              Process                |     335 |       1372160 |       1.31 |                  n/a
              Module                 |       0 |             0 |       0.00 |                  n/a
              Map/Shared             |       0 |             0 |       0.00 |                  n/a

   Program memory that would be left if paging pool pages above 'target' were freed:
   Program Memory                    |   28878 |     118284288 |     112.80 |                  n/a
     AVAILABLE PROGRAM MEMORY        |   26441 |     108302336 |     103.29 |                  n/a
     USED PROGRAM MEMORY             |    2437 |       9981952 |       9.52 |                  n/a
       Kernel Objects                |      91 |        372736 |       0.36 |                  n/a
       Unused Paging Pool            |     794 |       3252224 |       3.10 |                  n/a
       Unaccounted / Unknown         |     742 |       3039232 |       2.90 |                  n/a
       Programs                      |    1604 |       6569984 |       6.27 |                  n/a

   Paging Pool Reference Values (actual RAM consumed is already counted above)
   Loader Pool (Current In Use)      |     124 |        507904 |       0.48 |                  n/a
   Loader Pool (Current Free)        |     644 |       2637824 |       2.52 |                  n/a
   Loader Pool (Current Consumption) |     768 |       3145728 |       3.00 |                  n/a
   Loader Pool (Target)              |     768 |       3145728 |       3.00 |                  n/a
   Loader Pool (Maximum)             |    2048 |       8388608 |       8.00 |                  n/a

   File Pool (Current In Use)        |     106 |        434176 |       0.41 |                  n/a
   File Pool (Current Free)          |     150 |        614400 |       0.59 |                  n/a
   File Pool (Current Consumption)   |     256 |       1048576 |       1.00 |                  n/a
   File Pool (Target)                |     256 |       1048576 |       1.00 |                  n/a
   File Pool (Maximum)               |    2560 |      10485760 |      10.00 |                  n/a

   Loader Paging Pool (fail count)             |
   Loader Paging Pool (trim count)             |             0
   Loader Paging Pool (critical count)         |             0

   File Paging Pool (fail count)               |             0
   File Paging Pool (trim count)               |             0
   File Paging Pool (critical count)           |             0
```

System Memory Map Report

The DevHealth System Memory Map Report provides detailed information about the virtual memory usage of the device at the moment that you run DevHealth. This report, which lists page types for all processes, provides similar data to that of the Visual Studio 2008 Target Control command `mi full`. However, the System Memory Map Report that DevHealth provides contains more detailed information, such as the virtual memory address range that each DLL module uses. The System Memory Map Report of each process contains the base address of a virtual memory chunk, a symbol that represents a single page of virtual memory, and a summary of the information for the component that uses the corresponding pages.

The meanings of the symbols are listed in the table below.

Symbol	Meaning
- (hyphen)	Reserved but not in use. This symbol indicates a virtual page that is currently allocated but is not mapped to any physical memory.
C	Executable code either in ROM or in the NK region outside of system RAM.
c	Executable code in RAM.
R	Read-only data section in an .exe or .dll file either in ROM or in the NK region outside of system RAM. May also be a memory-mapped file in ROM.
r	Read-only data section in RAM, or RAM committed as read-only. If this read-only data is not an .exe or a .dll file, it is most likely a memory-mapped file.
D	Writable data section in a .dll file. Cannot be shared between processes.
E	Writable data section in an .exe file.
p	Paging pool.
S	Committed stack page for a thread.
H	Heap.
W	RAM committed as read-write. Most likely

Symbol	Meaning
	mapped memory.
P	A peripheral that is not part of system RAM. The peripheral may be video memory, a camera, or some other device.
d	Duplicate. This page of RAM was already accounted for. For example, it might be a shared module or an address that was virtually copied. Assignment of duplicate addresses follows the order of DevHealth output.

An excerpt from a System Memory Map Report is shown below. This excerpt shows the memory usage of one process. The System Memory Map Report of each process contains three columns. Each line represents 64 KB of virtual memory in the process. The first column represents the base address of the 64 KB virtual memory chunk. (When an address is skipped, it signifies that no pages are presently allocated in the entire 64 KB chunk.) The second column contains a symbol that represents a single page of virtual memory (see the table above for an explanation of the symbols). There are 16 symbols because each page is 4 KB. The third column of the report summarizes the information for the component that uses the corresponding pages.

```
Memory usage for Process d21347b0: 'explorer.exe' pid 3710012

Block base 00000000 end 00054000

  00000000: ----- ----------
  00010000: -CCCCCCCCCCCCCCC
  00020000: CCCCCCCCCCCCCCCC
  00030000: CCW--CCCCCCCCCCC
  00040000: CCCCCCCCCCCCCCCC
  00050000: CCCC
  00060000: -------------SSS
  00070000: HHHHHH---------   <-- Heap:0x00070010
                              === File hMap=0x9FF62858 "NLSFILE" (0x000802D4 - 0x000B31A1)
  00080000: dddddddddddddddd
  00090000: dddddddddddddddd
  000a0000: dddddddddddddddd
  000b0000: dddd
                              === end hMap
  000c0000: ---------------S
  000d0000: HHHHHHHHHHHH---   <-- Heap:0x000d0010
  000e0000: --------------SS
  000f0000: ---------------S
  ...
  40000000: ----------------
                              === coredll.dll (40010000 - 400a9fff)
  40010000: -ddddddddddddddd
  40020000: dddddddddddddddd
  40030000: dddddddddddddddd
  40040000: dddddddddddddddd
  40050000: dddddddddddddddd
  40060000: dddddddddddddddd
  40070000: dddddddddddddddd
  40080000: dddddddddddddddd
  40090000: dddddddD------dd  <-- DLL: coredll.dll
  400a0000: ddddd-----
                              === coredll.dll page summary: code=0[rom(C):0 ram(c):0] data r/o(R)=0
                                  r/w=1[ro(r)=0 rw(W)=0 exe(E)=0 dll(D)=1 heap(H)=0]
                                  page(p)=0 stack(S)=0 dup(d)=141 unknown(?)=0 obj(O)=0 peripheral(P)=0
                                  reserved(-)=12
                              === fpcrt.dll (400b0000 - 400c1fff)
  400b0000: -ddddddddddddddD  <-- DLL: fpcrt.dll
  400c0000: --
                              === fpcrt.dll page summary: code=0[rom(C):0 ram(c):0] data r/o(R)=0
                                  r/w=1[ro(r)=0 rw(W)=0 exe(E)=0 dll(D)=1 heap(H)=0]
                                  page(p)=0 stack(S)=0 dup(d)=14 unknown(?)=0 obj(O)=0 peripheral(P)=0
                                  reserved(-)=3
                              === notify.dll (400d0000 - 400dcfff)
                              === notify.dll page summary: code=0[rom(C):0 ram(c):0] data r/o(R)=0
                                  r/w=0[ro(r)=0 rw(W)=0 exe(E)=0 dll(D)=0 heap(H)=0]
                                  page(p)=0 stack(S)=0 dup(d)=0 unknown(?)=0 obj(O)=0 peripheral(P)=0
                                  reserved(-)=0
  ...
                              === zlib.dll (40100000 - 4010cfff)
  40100000: -CCCCCCCCCD--     <-- DLL: zlib.dll
                              === zlib.dll page summary: code=9[rom(C):9 ram(c):0] data r/o(R)=0
                                  r/w=1[ro(r)=0 rw(W)=0 exe(E)=0 dll(D)=1 heap(H)=0]
                                  page(p)=0 stack(S)=0 dup(d)=0 unknown(?)=0 obj(O)=0 peripheral(P)=0
                                  reserved(-)=3
  ...
                              === commctrl.dll (40120000 - 40184fff)
  40120000: -ddddddddddddddd
  40130000: dddddddddddddddd
...
```

Process Report

The DevHealth Process Report displays the number of pages of certain page types that each process uses at the moment that DevHealth runs. The page types that the Process Report accounts for are:

Symbol	Meaning
S	Committed stack page for a thread.
H	Heap.
E	Writable data section in an .exe file.
D	Writable data section in a .dll file. Cannot be shared between processes.
C	Executable code in RAM.
R	Read-only data section in RAM, or RAM committed as read-only. If this read-only data is not an .exe or a .dll file, it is most likely a memory-mapped file.
W	RAM committed as read-write. Most likely mapped memory.

An example of a Process Report is shown below, followed by an explanation of its columns.

```
Process            | PID         Base        Page? |  'S'  'H'  'E'  'D'  'c'  'r'  'W' |  Total
NK.EXE             | 0x00400002  0x00000000  N     |  134  475    0  272    0   81  317 |   1279
shell.exe          | 0x01240002  0x00000000  Y     |    1    1    0    2    0    0    3 |      7
udevice.exe        | 0x01bd0002  0x00000000  PART  |    9   11    0   21    0   25    1 |     67
udevice.exe        | 0x00d20006  0x00000000  PART  |    1    2    0    3    0   52    1 |     59
udevice.exe        | 0x01490006  0x00000000  PART  |    1    3    0   15    0   11    1 |     31
udevice.exe        | 0x03790002  0x00000000  PART  |    2    8    0    6    0    1    1 |     18
explorer.exe       | 0x03710012  0x00000000  Y     |    7   18    0   16    0    0    1 |     42
EmulatorStub.exe   | 0x04030002  0x00000000  Y     |    4    1    0    1    0    0    1 |      7
servicesd.exe      | 0x040e0002  0x00000000  Y     |    8   22    0   32    0    2    2 |     66
udevice.exe        | 0x04c30002  0x00000000  PART  |    1    2    0   10    0    1    1 |     15
devhealth60.exe    | 0x03c70006  0x00000000  CANNOT|    1    1    0    1    2    2    6 |     13
Total              |                                 |  169  544    0  379    2  175  335 |   1604
```

The meanings of the columns are listed below.

- The **Process** column shows the name of the process.
- The **PID** column shows the process ID (PID).
- The **Base** column is always zero in this report.
- The **Page?** column shows whether the module is pageable or not. The meanings of the entries in this column are given below.

Page? column entry	Description
Y	The module (.exe or .dll) is fully pageable.
N	The module is stored on media where it could be paged, but it is currently unpaged. This means that all of the code and data for the module is consuming RAM while the module is loaded. To make the module page-able, you would have to set linker settings to allow paging and/or set the module to be page-able in the .bib files or driver registry settings.
CANNOT	The module is stored on media where it cannot be paged (usually the Release Directory File System Drive (RELFSD)), so it is unpaged. This means that all of the code and data for the module is consuming RAM while the module is loaded.
PART	The module is stored on media where it could be paged, but only part of the module is page-able. This is usually true of drivers that have some ISR/IST or power management code which cannot safely be paged, while the rest of the driver could be paged. The unsafe parts of the driver are reported to the linker as un-page-able, while the rest is allowed to page in order to reduce RAM consumption.

- The **Total** column displays the sum of the **S**, **H**, **E**, **D**, **c**, **r** and **W** values in the preceding columns. This total does not include pages of type **C**, **R**, **p**, **d**, **?**, **P**, and **-**, which are listed in the page summaries of the individual processes in the System Memory Map Report. Note that the Process Report summarizes the number of pages for certain page types, whereas the System Memory Map shows the layout.

Module Report

The DevHealth Module Report displays the number of pages for three page types for all DLL modules that are loaded in memory at the moment that DevHealth runs. The Module Report accounts for the following page types:

Page type	Description
c	Executable code in RAM.
r	Read-only data section in RAM, or RAM committed as read-only. If this read-only data is not an .exe or a .dll file, it is most likely a memory-mapped file.
D	Writable data section in a .dll file. Cannot be shared between processes.

An example of a Module Report is shown below, followed by an explanation of its columns.

```
Module                        | Base        End         Page?   |  'c'  'r'  'D' |  Total
 devhealthdll60.dll           | 0xd1850000  0xd18defff  CANNOT  |    0   12  129 |    141
 k.toolhelp.dll               | 0xc0380000  0xc0385fff  Y       |    0    0    1 |      1
 k.coredll.dll                | 0xc0090000  0xc0127fff  PART    |    0    6    1 |      7
 k.fpcrt.dll                  | 0xc0130000  0xc0141fff  PART    |    0    1    1 |      2
 gwes.dll                     | 0xc01c0000  0xc0293fff  Y       |    0    0    6 |      6
 k.iphlpapi.dll               | 0xc04b0000  0xc04bffff  Y       |    0    0    1 |      1
 ipv6hlp.dll                  | 0xc06f0000  0xc06fbfff  Y       |    0    0    2 |      2
 ne2000.dll                   | 0xc0660000  0xc0668fff  Y       |    0    0    1 |      1
 ndis.dll                     | 0xc05a0000  0xc05c3fff  Y       |    0    0    1 |      1
 commctrl.dll                 | 0x40120000  0x40184fff  Y       |    0    0    0 |      0
 coredll.dll                  | 0x40010000  0x400a9fff  Y       |    0    0    0 |      0
 ssllsp.dll                   | 0x402a0000  0x402aafff  Y       |    0    0    0 |      0
 wspm.dll                     | 0x40270000  0x40275fff  Y       |    0    0    0 |      0
 netui.dll                    | 0x40310000  0x4034cfff  Y       |    0   16    0 |     16
…

                                           Pages       Size (MB)
Total Code for all Modules:                    0          0.00
Total Read for all Modules:                  168          0.66
Total Data for all Modules:                  272          1.06
==================================================================
Total for all Modules:                       440          1.72

                                           Pages        Size (MB)
Total Code for non-pageable Modules:           0          0.00
Total Read for non-pageable Modules:          44          0.17
Total Data for non-pageable Modules:         213          0.83
==================================================================
Total for non-pageable Modules:              257          1.00
```

The meanings of the columns are listed below.

- The **Module** column shows the name of the module.
- The **Base** column is the starting address of the module in virtual memory.
- The **End** column is the ending address of the module in virtual memory.
- The **Page?** column shows whether the module is pageable or not. The meanings of the entries in this column are given below.

Page? column entry	Description
Y	The module (.exe or .dll) is fully pageable.
N	The module is stored on media where it could be paged, but it is currently unpaged. This means that all of the code and data for the module is consuming RAM while the module is loaded. To make the module page-able, you would have to set linker settings to allow paging and/or set the module to be page-able in the .bib files or driver registry settings.
CANNOT	The module is stored on media where it cannot be paged (usually the Release Directory File System Drive (RELFSD)), so it is unpaged. This means that all of the code and data for the module is consuming RAM while the module is loaded.
PART	The module is stored on media where it could be paged, but only part of the module is page-able. This is usually true of drivers that have some ISR/IST or power management code which cannot safely be paged, while the rest of the driver could be paged. The unsafe parts of the driver are reported to the linker as un-page-able, while the rest is allowed to page in order to reduce RAM consumption.

- The **Total** column displays the sum of the **c**, **r**, and **D** values in the preceding columns.

Heap Report

For each heap in a process, the DevHealth Heap Report provides the following information:

- The start and end address of the heap region
- The number of allocation blocks
- The total size of the allocation

If the heap sentinel is enabled, the Heap Report section labeled **Per-module consumption** shows the number of allocation blocks and the size of total allocations by each module. To enable the heap sentinel in the OS image, set **IMGENABLEHEAPSENTINEL=1**. The **Per-module consumption** section is labeled as "Allocated by unknown modules" if the heap sentinel is not enabled.

Below is an excerpt of a Heap Report, which shows that there are two heaps in process explorer.exe: A heap starting at address 0x00070010 and a heap starting at address 0x000d0010. In the first heap, five modules (ceshell.dll, explorer.exe, commctrl.dll, coredll.dll, and ole32.dll) have allocated memory and the total allocation size is 14214 bytes.

```
Process:    explorer.exe      Heaps: 2
           Heap: 0x00070010   Size:         16   Alloc:      14214
             Regions:
                 170 blocks   address:     0x00070000 - 0x00076000
             Per-module consumption:
                  33 blocks    2262 bytes  ceshell.dll
                  22 blocks    3820 bytes  explorer.exe
                  27 blocks    2558 bytes  commctrl.dll
                  52 blocks    3530 bytes  coredll.dll
                   7 blocks    2044 bytes  ole32.dll
                 141 blocks   14214 bytes  TOTAL ALLOC
                  29 blocks    6048 bytes  FREE
           Heap: 0x000d0010   Size:         16   Alloc:       1484
             Regions:
                   8 blocks   address:     0x000d0000 - 0x000dc000
             Per-module consumption:
                   5 blocks     888 bytes  imaging.dll
                   1 blocks     596 bytes  coredll.dll
                   6 blocks    1484 bytes  TOTAL ALLOC
                   2 blocks   47488 bytes  FREE

           Total per-module consumption for all heaps in explorer.exe:
                   5 blocks     888 bytes  imaging.dll
                  53 blocks    4126 bytes  coredll.dll
                  33 blocks    2262 bytes  ceshell.dll
                  22 blocks    3820 bytes  explorer.exe
                  27 blocks    2558 bytes  commctrl.dll
                   7 blocks    2044 bytes  ole32.dll
                 147 blocks   15698 bytes  TOTAL ALLOC
                  31 blocks   53536 bytes  FREE
 End process explorer.exe
```

The Heap Report also contains total heap allocations across the entire system on a module basis. The following excerpt shows heap report data across the entire system.

```
          Total per-module consumption across entire system:
                  1 blocks      16 bytes  devhealth60.exe
                367 blocks   25798 bytes  coredll.dll
                  3 blocks     668 bytes  udevice.exe
                 39 blocks    5144 bytes  dcomssd.dll
                  6 blocks     208 bytes  iphlpapi.dll
                  7 blocks     656 bytes  ntlmssp_svc.dll
                  4 blocks     676 bytes  udevice.exe
                  3 blocks    1634 bytes  netui.dll
                  4 blocks      64 bytes  softkb.dll
                  1 blocks      12 bytes  largekb.dll
                  2 blocks     434 bytes  notify.dll
                794 blocks   59628 bytes  devhealthdll60.dll
                454 blocks  139708 bytes  k.coredll.dll
               1418 blocks  280208 bytes  gwes.dll
                 22 blocks    1464 bytes  ipv6hlp.dll
                  6 blocks     208 bytes  k.iphlpapi.dll
                 36 blocks    1492 bytes  tcpstk.dll
                590 blocks  454600 bytes  cxport.dll
                984 blocks   67996 bytes  filesys.dll
                 98 blocks   67348 bytes  afd.dll
                 38 blocks   17916 bytes  audevman.dll
                  6 blocks     408 bytes  s3c2410x_wavedev.dll
                 12 blocks    5504 bytes  serial_smdk2410.dll
                286 blocks   30512 bytes  busenum.dll
                  2 blocks    2104 bytes  filterfsd.dll
…
               6774 blocks 3541316 bytes  TOTAL ALLOC
                637 blocks  384128 bytes  FREE
```

Dependency Report

The DevHealth Dependency Report cross-references the reference counts between running processes and loaded DLL modules. The Dependency Report contains two sections. The first section lists, for each process, the number of references that the process has to each DLL module that the process is using. The second section shows this information in the opposite way: for every loaded DLL, it lists which running processes have references to that DLL.

An excerpt of a Dependency Report is shown below (many lines of content were removed for readability).

```
Dependency Report
=================
Process module dependencies

 NK.EXE
   - devhealthdll.dll          Ref: 2
   - gwes.dll                  Ref: 2
   - k.coredll.dll             Ref: 109
   - k.toolhelp.dll            Ref: 1
   - ceshell.dll               Ref: 1
   - urlmon.dll                Ref: 1
   - k.iphlpapi.dll            Ref: 3

 shell.exe
   - locale.dll                Ref: 1
   - normalize.dll             Ref: 1
   - dllheapinfoext.dll        Ref: 1
   - toolhelp.dll              Ref: 1
   - coredll.dll               Ref: 1
   - relfsdext.dll             Ref: 1

 udevice.exe
   - locale.dll                Ref: 1
   - normalize.dll             Ref: 1
   - softkb.dll                Ref: 1
   - rpcrt4legacy.dll          Ref: 1
   - lpcrt.dll                 Ref: 1
   - coredll.dll               Ref: 1
   - largekb.dll               Ref: 1
   - bcrypt.dll                Ref: 1

Reference Counts
                           | NK.EXE | shell.exe | udevice.exe | explorer.exe | compositor.exe |
devhealthdll.dll           | 1      |           |             |              |                |
gwes.dll                   | 2      |           |             |              |                |
ceshell.dll                | 1      |           |             |              |                |
kernel.dll                 | 1      |           |             |              |                |
toolhelp.dll               | 1      | 1         |             |              |                |
dllheapinfoext.dll         | 1      | 1         |             |              |                |
relfsdext.dll              | 1      | 1         |             |              |                |
softkb.dll                 | 1      |           | 1           |              |                |
ntlmssp_svc.dll            | 1      |           |             |              |                |
notify.dll                 | 1      |           |             |              |                |
credprov.dll               | 1      |           |             |              |                |
ws2serv.dll                | 1      |           |             |              |                |
gdicompositor.dll          | 1      |           |             |              | 1              |
ddraw.dll                  | 1      |           |             |              | 1              |
servicesfilter.dll         | 1      |           |             |              |                |
lpcd.dll                   | 1      |           |             |              |                |
winsock.dll                | 1      |           |             |              |                |
commctrl.dll               | 2      |           |             | 1            |                |
console.dll                | 1      |           |             | 1            |                |
locale.dll                 | 10     | 1         | 1           | 1            | 1              |
```

Conclusion

By using the DevHealth tool, you can obtain an overview of the memory usage of a Windows Embedded Compact 7 device at the moment that you run the tool. DevHealth produces several types of reports that show memory usage from multiple perspectives: from the amount of physical memory on the device and how that memory is being consumed to the number and type of virtual memory pages that are being used by each process and DLL module. DevHealth also provides information on heap usage and reference counts for all processes and modules.

DevHealth is not designed to replace other Windows Embedded Compact 7 memory analysis tools. However, it is easy to run from a Command Prompt window on your device without the need for a connection to your development computer. Or, with a connection to your development computer, you can run it from Platform Builder by using Target Control. DevHealth automatically saves its report as a text file that you can read by using any text file viewer. Depending on the command-line options you choose, the report can contain a variety of information that you can use to monitor memory usage and as a start for memory issue investigations.

Additional Resources

Windows Embedded website (http://go.microsoft.com/fwlink/?LinkId=183524)

Remote Tools in Platform Builder (http://go.microsoft.com/fwlink/?LinkId=238314)