

# Expand The Memory Of Your 32-Bit Application Using Microsoft's 4GT

**Uwe Kranich**

**ADVANCED MICRO DEVICES, INC.  
One AMD Place  
Sunnyvale, CA 94088**

**June 2, 2004**

## Abstract

The 4GB memory limit of 32-bit computing, for many applications, results in a performance penalty since effective data processing requires sufficient addressable memory. 64-bit processing, using AMD64 technology, enables applications to gain access to a much larger memory space and also provides additional performance gains due to the larger number of CPU registers.

Applications that demand expanded memory requirements are ideal first candidates for 64-bit porting. Data-intensive applications; graphic-intensive design tools; visualization; and media applications are among these best candidates.

A full 64-bit port is sometimes too large a first step, especially when it demands a serious redesign of the application's software architecture. This is especially true for media applications as they often depend on an ecosystem of 3<sup>rd</sup> party plugins to provide additional functionality and applications that build using 3<sup>rd</sup> party static libraries. In these cases the 64-bit application will only be able to provide the same level of functionality as the 32-bit counterpart if a major percentage of the plugins and/or libraries are also available in 64-bit.

The following article describes a simple, first step migration path for Windows-based applications that enable existing 32-bit applications to increase available virtual memory to 4GB thus easing the evolution into 64-bit application development.

This method, termed by Microsoft as 4GT, may be used by any 32-bit Windows application and provides an immediate benefit of a larger pool of virtual memory through enabling a "Large Address Aware" (LAA) flag in the executable binary. As mentioned above, it is particularly beneficial for 32-bit memory intensive applications, such as those found in media, since it increases the amount of available virtual memory while still being able to use existing plugins and dll's. Links to Microsoft articles are in Appendix A.

## Contents

Abstract .....	2
Contents .....	3
Introduction.....	4
Microsoft 4GT .....	6
Large Address Aware ("LAA") Implementation .....	8
Modifying the LAA Bit .....	8
Determine the state of the LAA bit in the binary.....	9
Development considerations.....	10
Windows 64 Registry and File location characteristics.....	11
Registry entries.....	11
Program File locations .....	12
System and dll locations .....	12
32-bit Application Impact.....	12
Summary .....	12
Appendix A: Microsoft Developer resources.....	14
AMD Overview .....	15

## Introduction

Under 32-bit Windows® XP, most 32-bit applications can only use up to 2 GB virtual memory. This is limited by the fact that the operating system kernel allocates 2GB of virtual memory for its own usage. The 2GB user space and the 2GB kernel space totals to 4GB, the maximum address range of a 32-bit processor.

Windows XP also allows a system to boot with the "/3GB" cmd line switch. In that mode, the operating system allocates 3GB of virtual memory for the user space and limits the kernel to 1GB. This mode has some restrictions in the number of page table entries (PTE's) combined with the Paged Pool Area (see [MS\_4GT\_3]).

An application can only use 3GB if the executable file has the "IMAGE\_FILE\_LARGE\_ADDRESS\_AWARE" flag (In future called: "LAA" flag) enabled. This is a linker option or can be changed with some tools, which can modify existing binaries.

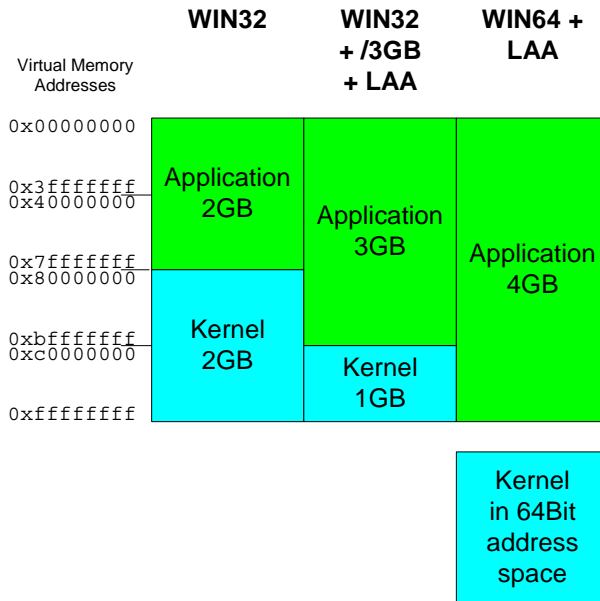
A major advantage of a 64-bit operating system is its ability to address much more than 4GB virtual memory. In a 64-bit operating environment, all 64-bit applications can take advantage of memory space in excess of the 4GB limit found in 32-bit operating environments. 32-bit applications running on 64-bit Windows can also benefit from the underlying 64-bit Windows architecture. 64-bit Windows can allocate 4GB virtual memory for each 32-bit process when the LAA flag is enabled. This is due to the fact, that the kernel resides in the 64-bit memory space and, as a result, is not limiting the 32-bit user space. In addition, this ability doesn't have the above mentioned limitation of PTE's.

Leveraging the LAA flag, existing 32-bit applications can be changed to allow access to 4GB of virtual memory when running under a 64-bit version of Windows. This allows the 32-bit application to get an additional boost in virtual memory while still being able to use all existing 32-bit static libraries and plugins. In most cases, enabling this functionality ONLY requires the LAA bit to be enabled in the executable. Executables with the LAA flag enabled are "backwards" compatible, so only one binary is required for 32-bit and 64-bit Windows (More details in section 3).

64-bit Windows does not require any special boot option (like "/3GB") and it will always allow 32-bit applications to allocate 4GB if the LAA flag is enabled.

Figure 1.1 shows the different virtual memory allocations.

**Figure 1.1 Different Virtual memory allocation**



## Microsoft 4GT

4GT allows a 32-bit application a straightforward migration to 64-bit Windows and has the following attributes:

- 4GB virtual memory per 32-bit application.
- Each 32-bit application (process) can use a maximum of 4GB virtual memory.

### Common FAQs

Q: Are there any other changes necessary to take advantage of 4GB virtual memory?

A: No, only the LAA bit needs to be set with either a linker or binary change. No other source code and binary code changes are required.

Q: Can I use all my existing 32-bit tools?

A: Existing 32-bit tools can be used to continue to build all your 32-bit applications and debug them.

Q: What about existing 32-bit static libs?

A: Existing 32-bit static libs can be used to build the application. As an example some applications use 3rd party image or sound libs.

Q: How about existing 32-bit plugins for media applications?

A: This is probably the largest benefit for media applications. Existing 32-bit plugins, which consist primarily of dll's, can be used immediately.

Q: Is there only one 32-bit binary?

A: Yes, a single binary supports both 32-bit and 64-bit Windows.

Q: Does this require the use of 64-bit device drivers?

A: Yes, 64-bit Windows requires 64-bit device drivers. This is an opportunity to become familiar with 64-bit device drivers and the 64-bit Windows environment with a known working 32-bit application.

Q: Does it support extended registers?

A: The additional 8 integer and 8 SSE registers of the AMD64 architecture only can be used in a 64-bit application. The benefits of the additional registers are not available to an "LAA" 32-bit application.

Q: What about code and libraries?

A: The LAA functionality enablement can be used immediately with code and libraries which handle full 32-bit pointers correctly (see section 3.3). Any code or libraries that do not handle 32-bit pointers correctly or have hardcoded memory limits will need to have some additional reprogramming to utilize this method.

Table 2.1 summarizes the properties of a normal 32-bit application, an LAA 32-bit application and a full 64-bit application using 64-bit Windows.

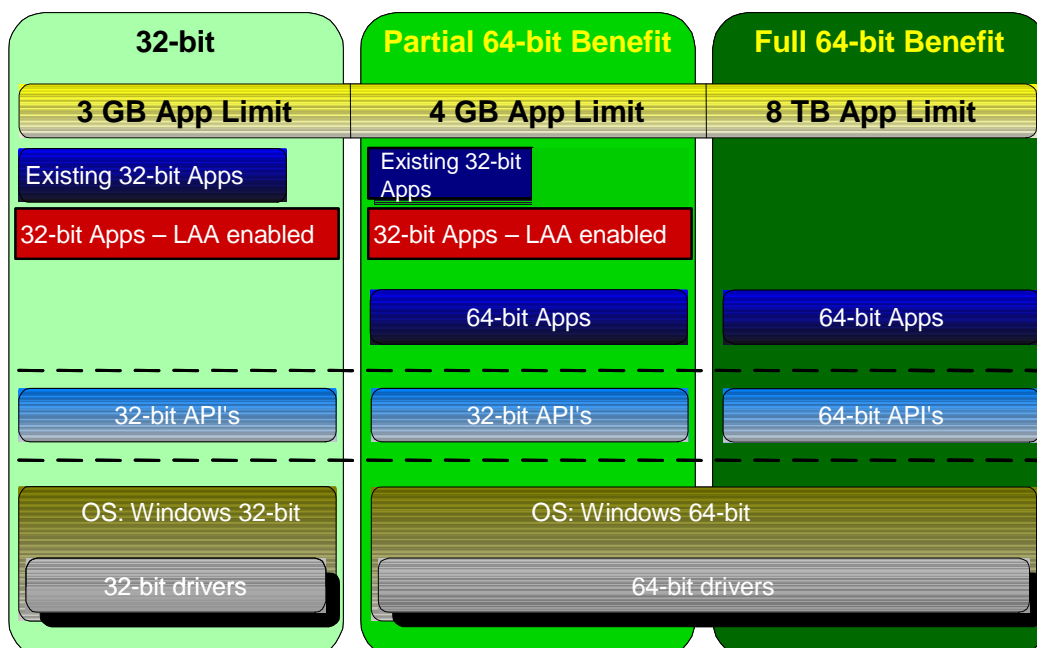
**Table 2.1 Property matrix**

Property	32-bit	32-bit-LAA	64-bit
Address space	2GB	3-4GB	>4GB
Use existing 32-bit tools	Y	Y	N
Use existing 32-bit static libs to build application	Y	Y	N
Use existing 32-bit dll's and plugins	Y	Y	N
Requires 64-bit Windows and AMD64 architecture	N	Y	Y
Requires 64-bit device drivers	N	Y	Y
Extended registers of AMD64 architecture	N	N	Y

**Figure 2.1: Application Evolution**

Figure 2.1 below illustrates the application benefit when migrating from a 32-bit application to a full 64-bit port. The top row of the figure represents the different available memory sizes in the associated columns. As figure 2.1 represents in the '32-bit' column, an existing 32 application without LAA enabled can only leverage up to 2GB. However, when LAA is enabled in a 32-bit operating environment, the same application can leverage up to 3 GB of available memory.

Additionally, in the 'Partial 64-bit Benefit' column, an existing 32 bit application can leverage up to 2GB of the 4 GB available application memory limit when running in a 64-bit operating system environment. When the LAA option is enabled on a 32-bit application in this instance, it can leverage up to 4 GB in a 64-bit operating system environment.



## Large Address Aware ("LAA") Implementation

As described in the introduction the LAA flag needs to be enabled in the binary to allow 64-bit Windows to allocate more than 2GB virtual memory for each 32-bit process. If this bit is not enabled, then 64-bit Windows only will allocate a maximum of 2GB and an application will still have the same virtual memory limitations as it does when running under 32-bit Windows.

The LAA bit is just an indication to the OS loader that the 32-bit application (process) supports addresses larger than 2GB. Once the LAA bit option is enabled, it is then up to the OS to allocate more than 2GB. In a standard 32-bit Windows configuration, the operating system will allocate a 32-bit application using the LAA option a maximum of 3GB. In the 64-bit Windows environment, the operating system will allocate a maximum of 4GB.

Therefore, only ONE binary is required (with the LAA flag enabled) to leverage this additional memory space allocation. Through the use of that binary, you can transparently run 32-bit or 64-bit Windows versions and take advantage of the added benefit of making use of the 4GB virtual memory space under 64-bit Windows.

The LAA change to your application may be easily integrated into a point release, service pack or update of an application. Since the source code and binary code are not modified (just the flag is changed), the change should require minimal additional testing and validation.

### Modifying the LAA Bit

There are basically 2 different methods for changing the LAA bit:

#### 1. Change the Linker options

Add the "/LARGEADDRESSAWARE" switch to the command line. This instructs the linker to set the LAA bit in the binary.

Visual Studio .NET 2003 has that flag also as a menu item in the "Linker\System" settings of the project properties.

#### 2. Modify binary

All Visual Studio implementations contain the "editbin.exe" command line tool in the corresponding "bin" directory.

editbin allows you to set or reset the LAA bit in the binary file (executable). This is done with the /LARGEADDRESSAWARE option. Here are 2 command line examples on how to set and reset the LAA bit in the binary file called "test.exe"

```
Set LAA:    "editbin /LARGEADDRESSAWARE test.exe"
```

```
Reset LAA: "editbin /LARGEADDRESSAWARE:NO test.exe"
```

As can be seen neither method modifies the source code, nor the binary. Just the LAA bit is enabled.

## Determine the state of the LAA bit in the binary

To check, whether the binary has the LAA bit enabled, the "dumpbin.exe" command line tool can be used. This is also part of all Visual Studio implementations and is also located in the corresponding "bin" directory. The following command line displays the LAA bit of the binary file "test.exe":

```
"dumpbin /headers test.exe"
```

This generates a large output with all the associated sections. Immediately at the beginning, there is a list of the "FILE HEADER VALUES". If the "FILE HEADER VALUES" list contains a statement such as:

```
"Application can handle large (>2GB) addresses"
```

then the application has the LAA bit enabled. If this line is not existent, then the LAA bit is not enabled.

Two sample outputs show the LAA bit enabled and not enabled.

LAA bit is enabled in test.exe below.

```
Microsoft (R) COFF/PE Dumper Version 7.10.3077
Copyright (C) Microsoft Corporation. All rights reserved.

Dump of file test.exe

PE signature found

File Type: EXECUTABLE IMAGE

FILE HEADER VALUES
    14C machine (x86)
      3 number of sections
3E77E638 time date stamp Wed Mar 19 04:38:32 2003
      0 file pointer to symbol table
      0 number of symbols
      E0 size of optional header
    12F characteristics
          Relocations stripped
          Executable
          Line numbers stripped
          Symbols stripped
          Application can handle large (>2GB) addresses
          32 bit word machine
```

LAA bit is not enabled in test.exe below.

```
Microsoft (R) COFF/PE Dumper Version 7.10.3077
Copyright (C) Microsoft Corporation. All rights reserved.

Dump of file test.exe

PE signature found

File Type: EXECUTABLE IMAGE

FILE HEADER VALUES
    14C machine (x86)
      3 number of sections
    3E77E638 time date stamp Wed Mar 19 04:38:32 2003
      0 file pointer to symbol table
      0 number of symbols
    E0 size of optional header
    12F characteristics
      Relocations stripped
      Executable
      Line numbers stripped
      Symbols stripped
      32 bit word machine
```

### Development considerations

The following considerations will help to make sure that the code can handle addresses larger than 2GB:

- Avoid the use of signed pointer arithmetic (i.e. compares and adds)
- Pointers use all 32-bits. Don't use Bit31 for something else.
- Some dll's will be loaded just under the 2GB boundary. In this case, no consecutive memory can be allocated with VirtualAlloc().
- Whenever possible, use GlobalMemoryStatusEx() (preferred) or GlobalMemoryStatus() to retrieve memory sizes.

Because most software developers typically avoid these types of situations, in many cases LAA can be leveraged without any additional coding.

If, however, an application has any of the above issues, additional programming work will be required. Such programming effort would still be considered worthwhile since it will generate 32-bit clean code that can have the added benefit of accessing 4GB of virtual memory under 64-bit Windows. Also, this clean code is a safe foundation for full 64-bit porting.

## Windows 64 Registry and File location characteristics

This section describes how 64-bit Windows handles the coexistence of 32-bit and 64-bit applications and is provided as background information only.

Since the operating system uses redirection services, the information below should not impact most 32-bit applications under 64-bit Windows as it is primarily transparent to the 32-bit applications.

64-bit Windows maintains separate locations for 32-bit and 64-bit applications in the registry as well as in the file system. In addition, 64-bit Windows has separate locations for 32-bit and 64-bit system files and dll's. In principle, all 64-bit applications and dll's reside in the same locations as in 32-bit Windows. A new location is used for 32-bit applications.

The sections below describe these issues in more detail.

### Registry entries

64-bit Windows requires different registry entries for 32-bit and 64-bit applications. Therefore 32-bit and 64-bit applications have a different registry path for following software related entries:

- "HKEY\_LOCAL\_MACHINE\SOFTWARE"
  - For 64-bit applications this is the original path:  
"HKEY\_LOCAL\_MACHINE\SOFTWARE\"
  - For 32-bit applications this is now the path:  
"HKEY\_LOCAL\_MACHINE\SOFTWARE\Wow6432Node\"
  
- "HKEY\_CLASSES\_ROOT"
  - For 64-bit applications this is the original path:  
"HKEY\_CLASSES\_ROOT"
  - For 32-bit applications this is now the path:  
" HKEY\_CLASSES\_ROOT\Wow6432Node\"

This difference is not visible to the 32-bit application. All registry references from a 32-bit application to:

```
"HKEY_LOCAL_MACHINE\SOFTWARE"
```

automatically get redirected to:

```
"HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\"
```

by Windows. 64-bit applications simply access the registry with an unchanged path.

However, when regedit is used the difference is important. The usage of "\*.reg" or "\*.inf" files, which modify the registry directly during install or later, have to be carefully considered.

## Program File locations

64-bit Windows maintains 2 separate directories for Program Files as follow:

- 64-bit applications are in "\Program Files"
- 32-bit applications are in "\Program Files (x86)"

## System and dll locations

64-bit Windows maintains 2 separate directories for system files as follow:

- 64-bit system files are in "\WINDOWS\system32\"
- 32-bit system files are in "\WINDOWS\SysWOW64\"

## 32-bit Application Impact

Almost all existing 32-bit applications are installed correctly and use the correct registry entries, because the Windows redirects all information. However, using "\*.reg" and "\*.inf" files requires a reroute of the registry path. Additionally, absolute file addressing to "\Program Files" will use the 64Bit path, not the 32Bit path, which is incorrect.

All installers support 64-bit Windows specifics now. As a result the overall impact here is minimal.

## Summary

LAA is a starting step for migrating an application into the 64-bit Windows environment. LAA provides the benefits of allowing a 32-bit application to exploit the advantage of running with 64-bit Windows by providing the application with access to 4GB of virtual memory.

ISVs can use this approach, as a first step, in their evolution towards a full 64-bit port by allowing them to quickly make available to their customers a 64-bit capable product.

A data-intensive or graphic-intensive application such as media applications enjoy added benefits of LAA as it enables them to gain additional virtual memory while still being able to use all existing plugins.

The LAA option provides ISVs with the added flexibility to determine if it is more advantageous to directly port their application from 32-bit to 64-bit or whether it is better to evolve from 32-bit to 64-bit via leveraging the LAA option. Such a

decision depends on the associated application, the source code status and the required ecosystem to build and use the application.

## Appendix A: Microsoft Developer resources

Microsoft has the following documents available to describe the LAA feature. Microsoft refers to it as “4GT”. In some documents Microsoft just refers to the server operating system to be able to support 4GT. However all the following Windows 32-bit operating systems support LAA, as shown in [MS\_4GT\_1]:

- Windows Server 2003 family
- Windows XP Professional Edition
- Windows 2000 Datacenter Server
- Windows 2000 Advanced Server
- Windows NT 4.0 Enterprise Edition

WIN64 (64-bit Windows) supports LAA, but as described above, with a total of 4GB rather than 3GB.

[MS\_4GT\_1]

[http://msdn.microsoft.com/library/default.asp?url=/library/en-us/memory/base/4qt\\_ram\\_tuning.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/memory/base/4qt_ram_tuning.asp)

[MS\_4GT\_2]

[http://msdn.microsoft.com/library/default.asp?url=/library/en-us/win64/win64/preparing\\_your\\_application\\_for\\_64\\_bit\\_windows.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/win64/win64/preparing_your_application_for_64_bit_windows.asp)

[MS\_4GT\_3]

[http://www.microsoft.com/resources/documentation/WindowsServ/2003/all/techref/en-us/Default.asp?url=/resources/documentation/WindowsServ/2003/all/techref/en-us/W2K3TR\\_4qt\\_intro.asp](http://www.microsoft.com/resources/documentation/WindowsServ/2003/all/techref/en-us/Default.asp?url=/resources/documentation/WindowsServ/2003/all/techref/en-us/W2K3TR_4qt_intro.asp)

## AMD Overview

AMD (NYSE:AMD) designs and produces innovative microprocessors, Flash memory devices, and low-power processor solutions for the computer, communications, and consumer electronics industries. AMD is dedicated to delivering standards-based, customer-focused solutions for technology users, ranging from enterprises and governments to individual consumers. For more information visit [www.amd.com](http://www.amd.com).

Author's Disclaimer and Copyright:  
© 2003 Advanced Micro Devices, Inc.  
All rights reserved.

The contents of this document are provided in connection with Advanced Micro Devices, Inc. ("AMD") products and technology. AMD makes no representations or warranties with respect to the accuracy or completeness of the contents of this publication and reserves the right to make changes to specifications and product and technology descriptions at any time without notice. No license, whether express, implied, arising by estoppel or otherwise, to any intellectual property rights is granted by this publication. AMD assumes no liability whatsoever, and disclaims any express or implied warranty, relating to its products including, but not limited to, the implied warranty of merchantability, fitness for a particular purpose, or infringement of any intellectual property right. AMD, the AMD Arrow logo, AMD Athlon, AMD Opteron, and combinations thereof and 3DNow! are trademarks of Advanced Micro Devices, Inc. Windows is a registered trademark of Microsoft Corporation. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.