这个清单提供编码和安装注意事项的简要列表，主要针对正在创建将在 64 位版本 Windows 操作系统上运行的驱动程序开发人员。

此信息适用于以下操作系统：  
Microsoft Windows Vista  
Microsoft Windows Server 2008  
64 位版本的 Microsoft Windows Server 2003  
Microsoft Windows XP

**本页内容**

[64 位 Windows 驱动程序入门](http://msdn.microsoft.com/zh-cn/windows/hardware/gg487358#e5) [64 位 Windows 驱动程序入门](http://msdn.microsoft.com/zh-cn/windows/hardware/gg487358#e5)  
[64 位 Windows 编程模型](http://msdn.microsoft.com/zh-cn/windows/hardware/gg487358#e1b) [64 位 Windows 编程模型](http://msdn.microsoft.com/zh-cn/windows/hardware/gg487358#e1b)  
[编码指南和移植问题](http://msdn.microsoft.com/zh-cn/windows/hardware/gg487358#e1h) [编码指南和移植问题](http://msdn.microsoft.com/zh-cn/windows/hardware/gg487358#e1h)  
[64 位 Windows 的驱动程序安装](http://msdn.microsoft.com/zh-cn/windows/hardware/gg487358#e3lac) [64 位 Windows 的驱动程序安装](http://msdn.microsoft.com/zh-cn/windows/hardware/gg487358#e3lac)  
[开发 64 位驱动程序的参考资料](http://msdn.microsoft.com/zh-cn/windows/hardware/gg487358#e6mac) [开发 64 位驱动程序的参考资料](http://msdn.microsoft.com/zh-cn/windows/hardware/gg487358#e6mac)

**64 位 Windows 驱动程序入门**

预期会有越来越多的领域采用 64 位系统，特别是在企业应用中。随着广泛的设备覆盖和针对 64 位系统的高质量驱动程序变得可用，这种应用将增长得更快。

设备支持 64 位系统的条件是：

* 确保设备寻址达到 64 位物理内存。
* 使用 64 位安全编程实践编写 Windows® 驱动程序，并使用 64 位编译器找出问题所属的领域：
  + 记住，[Windows Driver Foundation (WDF),](http://msdn.microsoft.com/zh-cn/windows/hardware/gg463268)[Windows Driver Model (WDM)](http://msdn.microsoft.com/en-us/library/aa490248.aspx) 和工具对于 32 位和 64 位 Windows 都一样。
  + 遵循编码指南，解决与 I/O 控制（IOCTL）、直接内存访问（DMA）和传统设备驱动程序接口（DDI）相关的驱动程序特定的问题。
  + 确保驱动程序支持即插即用和电源管理。

注意，不支持 32 位驱动程序：

* 内核模式驱动程序运行在与 64 位 Windows 相同的地址空间中。
* 适应大型系统的驱动程序可以为分页/未分页池和系统缓存使用较大的地址空间。

[返回页首](http://msdn.microsoft.com/zh-cn/windows/hardware/gg487358#top) [返回页首](http://msdn.microsoft.com/zh-cn/windows/hardware/gg487358#top)

**64 位 Windows 编程模型**

Windows 64 位版本由 Windows 编程模型和 Microsoft Win32® API 发展而来。Windows 32 位和 64 位平台基于相同的源代码：

* 现有的应用程序可以扩展以满足企业需求。
* 新的设计可以使用巨大的地址空间和内存。
* 支持现有的 32 位应用程序。

此外：

* 64 位 Windows 编程模型使用相同的 Win32 DDI 和 API。
* 在 LLP64（LongLong 和指针）数据模型中，只有指针扩展到 64 位。所有其他的基本数据类型（整型和长整型）的长度保持在 32 位。虽然 64 位指针需要适应具有高达 16 TB 虚拟内存的系统，但是大部分数据仍然在 32 位整数范围内。对于大部分应用程序，将默认整型大小改成 64 位只会浪费空间。
* 64 位 Windows 编程模型添加大小明确的新数据类型，以及匹配指针精度的新整数类型。类型列表请参见“数据类型”表。
* 指针长度为 64 位，但是整型和长整型数据类型仍然是 32 位。
* 大部分内存分配都是 64 位。
* CPU 掩码扩展到 64 位。
* I/O 请求和 Unicode 字符串的长度仍然是 32 位。

**数据类型**

| **类型名称** | **类型含义** |
| --- | --- |
| **固定宽度数据类型** |  |
| LONG32、INT32 | 32 位有符号 |
| LONG64、INT64 | 64 位有符号 |
| ULONG32、UINT32、DWORD32 | 32 位无符号 |
| ULONG64、UINT64、DWORD64 | 64 位无符号 |
| **指针精度数据类型** |  |
| INT\_PTR、LONG\_PTR | 有符号整型、 指针精度 |
| UINT\_PTR、 DWORD\_PTR | 无符号整型、 指针精度 |
| SIZE\_T | 无符号计数、指针精度 |
| SSIZE\_T | 有符号计数、 指针精度 |

**内存布局**

| **内存** | **基于 x64** | **基于 Intel Itanium** |
| --- | --- | --- |
| 用户地址范围 | 0x10000  0x000007FFFFFEFFFF | 0x10000  0x000006FBFFFEFFFF |
| 系统缓存 | 1 TB | 1 TB |
| 超出空间 | 512 GB | 16 GB |
| 系统地址空间（用于内核线程等） | 128 GB | 128 GB |
| 页大小 | 4K | 8K |
| 分页的池 | 128 GB | 128 GB |
| 未分页的池 | 在 Windows Vista 和早期系统上占物理内存的 40%，最高达 128 GB； 在 Windows Server 2008、Windows Vista SP1 和以后系统上占物理内存的 75%，最高达 128 GB | 在 Windows Vista 和早期系统上占物理内存的 40%，最高达 128 GB； 在 Windows Server 2008、Windows Vista SP1 和以后系统上占物理内存的 75%，最高达 128 GB |
| 物理内存地址 | 64 位 | 64 位 |

[返回页首](http://msdn.microsoft.com/zh-cn/windows/hardware/gg487358#top) [返回页首](http://msdn.microsoft.com/zh-cn/windows/hardware/gg487358#top)

**编码指南和移植问题**

**编码指南**

* 使用 Windows 64 位和 32 位安全数据类型。
* 检查所有指针使用，特别是指针运算。
* 删除内联汇编代码（使用内部或本机汇编代码）。
* 对于特定于 x64 的代码，使用：  
  **#if defined** (\_\_AMD64\_\_)  
  没有任何 \_\_X64\_\_ 定义可用。
* 对于特定于 Itanium 代码，使用：  
  **#if defined**(\_\_IA64\_\_)

**驱动程序的移植问题**

* 驱动程序应该支持 IOCTL 命令的 32 位和 64 位版本。  
  参见“IOCTL 支持”部分。
* 应该为只能寻址 32 位物理地址的硬件实现 DMA 支持。  
  参见“DMA 支持”部分。
* 指针、多态使用和对齐问题适用于所有驱动程序。  
  参见“指针、多态使用和对齐问题”部分。
* 如有可能，所有驱动程序都应该通过使用 WDF 正确地支持即插即用和电源管理。
  + 在 WDM 驱动程序中，从驱动中心模型过渡到设备中心模型；就是说，处理即插即用和电源管理 I/O 请求包（IRP）或者遵循微型端口指南。
  + 对于打印、扫描和相机，用户模式驱动程序必须是 64 位。
  + 不允许使用传统 API（针对 Windows NT® 4.0）。
* 对于 64 位微型端口：
  + 遵循 Windows Driver Kit (WDK) 中的微型端口特定的指南。
  + 记住 PAGE\_SIZE 的不同（基于 Itanium 的系统上为 8 K，x64 系统上为 4 K）。
  + 指针、多态使用和对齐问题适用于此处。
  + IOCTL、即插即用和 DMA 问题不适用于大部分微型端口。

**构建环境和工具**

* 用于 x64 的构建环境是 AMD64。
* 64 位工具和开发过程与 32 位类似：
  + Debugging Tools for Windows

使用 64 位版本的 Windows 调试工具能够调试在 64 位处理器上运行的 32 位和 64 位用户模式应用程序。使用这个工具包来调试应用程序和 Windows 64 位上的 Windows 32 位（Windows-32-bit-on-Windows-64-bit，WOW64）模拟器。

要调试 Intel Itanium 或 x64 处理器上的应用程序，请使用为特定处理器设计的工具包，可以在此处下载：[http://www.microsoft.com/whdc/devtools/debugging/install64bit.mspx](http://msdn.microsoft.com/zh-cn/windows/hardware/gg463012).

* + WDK 中的交叉编译器

WDK 中的 64 位构建环境包含一个 64 位编译器，可以用来构建和测试运行在基于 x64 的处理器上的 64 位驱动程序代码。参见 WDK 中的问题和限制（[http://msdn.microsoft.com/en-us/library/aa489554.aspx）](http://msdn.microsoft.com/en-us/library/aa489554.aspx).

* + 已检验版本

驱动程序开发人员使用 Windows 的已检验版本来识别和诊断操作系统级别的问题。已检验版本中的大部分检查用来确定参数或数据结构中的值是否在期望的特定范围内。详细信息请参见[http://www.microsoft.com/whdc/DevTools/tools/chkblds.mspx](http://msdn.microsoft.com/zh-cn/windows/hardware/gg463171).

* + Driver Verifier

这个工具测试和捕获正常操作时通常难以发现的许多情况。Driver Verifier 将验证驱动程序未进行非法的函数调用或引起系统崩溃。它可以识别内存破坏、错误处理的 IRP、无效的 DMA 缓存占用以及可能的死锁等情况。详细信息请参见<http://msdn.microsoft.com/en-us/library/ms792872.aspx>.

**BIOS、ACPI 和补丁注意事项**

* 基于 Itanium 的系统必须支持 ACPI 2.0 64 位表。
* 64 位 Windows 支持 GUID 分区表（GPT）磁盘。
* x64 系统上不允许 BIOS 回调。
* 要与基于 x64 的 Windows 操作系统兼容，驱动程序必须避免以下实践：
  + 修改系统服务表，例如与**KeServiceDescriptorTable**挂钩。
  + 修改中断调度表 (IDT)。
  + 修改全局描述符表 (GDT)。
  + 使用不是由内核分配的内核堆栈。
  + 修补内核的任何部分（仅限基于 AMD64 的系统）。

Windows 在 x64 平台上强制执行这些规则。任何这种修改都会导致错误检测。

**IOCTL 支持**

包含依赖于指针的类型的 IOCTL 结构在 64 位和 32 位应用程序中具有不同的大小。驱动程序在从 32 位线程发出的 IOCTL 和从 64 位线程发出的 IOCTL 之间必须有所不同。驱动程序基于发出方的位宽验证输入和输出缓冲区的长度。

驱动程序可以使用三种可能的解决方案中的任意一种来同时支持 32 位和 64 位调用者：

* 避免在 IOCTL 结构中使用指针类型。
* 使用 **IoIs32bitProcess()** API。
* 在 IOCTL 代码的 Function 代码字段中为 64 位调用者定义一个位。

要使用 **IoIs32BitProcess()**，在代码中定义一个 32 位 IOCTL 结构，然后确定发出进程是 32 位还是 64 位。

**头文件中的 IOCTL 结构：**

1. typedef **struct** \_IOCTL\_PARAMETERS {


5. PVOID   Addr;


9. SIZE\_T  Length;


13. HANDLE  Handle;


17. } IOCTL\_PARAMETERS, \*PIOCTL\_PARAMETERS;

**32 位 IOCTL 结构：**

1. //


5. // This structure is defined


9. // inside the driver source code


13. //


17. typedef **struct** \_IOCTL\_PARAMETERS\_32 {


21. VOID\*POINTER\_32  Addr;


25. INT32            Length;


29. VOID\*POINTER\_32  Handle;


33. } IOCTL\_PARAMETERS\_32, \*PIOCTL\_PARAMETERS\_32;

**使用 IoIs32BitProcess 的 32 位和 64 位 IOCTL 例子：**

1. #ifdef \_WIN64


5. **case** IOCTL\_REGISTER:


9. **if** (IoIs32bitProcess(Irp)) {


13. /\* If this is a 32 bit process \*/


17. params32 =






25. (PIOCTL\_PARAMETERS\_32)(Irp>AssociatedIrp.SystemBuffer);


29. **if**(irpSp->Parameters.DeviceIoControl.InputBufferLength <






37. **sizeof**(IOCTL\_PARAMETERS\_32)) {


41. status = STATUS\_INVALID\_PARAMETER;


45. } **else** {


49. LocalParam.Addr = params32->Addr;


53. LocalParam.Handle = params32->Handle;


57. LocalParam.Length = params32->Length;


61. /\* Handle the ioctl here \*/


65. status = STATUS\_SUCCESS;
66. Irp->IoStatus.Information = **sizeof**(IOCTL\_PARAMETERS);
67. }


71. } **else** { /\* 64bit process IOCTL \*/


75. **params** = (PIOCTL\_PARAMETERS)


79. (Irp->AssociatedIrp.SystemBuffer);


83. **if** (irpSp->Parameters.DeviceIoControl.InputBufferLength


87. < **sizeof**(IOCTL\_PARAMETERS)) {


91. status = STATUS\_INVALID\_PARAMETER;






99. } **else** {


103. RtlCopyMemory(&LocalParam, **params**,






111. **sizeof**(IOCTL\_PARAMETERS));


115. /\*  Handle the ioctl here \*/


119. status = STATUS\_SUCCESS;


123. }


127. Irp->IoStatus.Information = **sizeof**(IOCTL\_PARAMETERS);


131. }


135. **break**;

如果选择为 64 位调用者定义一个位，那么使用 Function 字段中的高位。

当前 IOCTL 码有 5 个字段，包括一个 11 位的 Function 字段：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Device Type (16) | Access (2) | Custom (1) | Function (11) | Method (2) |

新的 IOCTL 码使用 Function 字段中的高位来指示 64 位调用者：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Device Type (16) | Access (2) | Custom (1) | 64Bit (1) | Function (10) | Method (2) |

**DMA 支持**

* 使用 PHYSICAL\_ADDRESS 类型定义来访问物理地址。PHYSICAL\_ADDRESS 为 64 位长。
* 将所有 64 位视为一个有效的物理地址。不要忽略高端的 32 位。
* 使用内核模式驱动程序框架（KMDF）或 Windows DMA DDI 来确保在所有平台上正确运行。当使用 KMDF 或 Windows DMA 例程时，Windows 处理大小问题。
* 在合适的时候使用分散/聚集 DMA。
* 通过使用具备 64 位寻址能力的设备极大地提高性能。

**指针、多态使用和对齐问题**

指针大小在应用程序和驱动程序中一样。用于移植应用程序的指南在下列情况中也适用于驱动程序：

* 多态数据使用（与指针大小相关）。
* 对齐问题。
* 常量和宏运算。

本节讨论前两种情况。第三种情况在 MSDN® 文章[“超越 Windows XP：现在准备好迎接 64 位版本 Windows 的到来”中讨论。](http://msdn.microsoft.com/msdnmag/issues/01/11/xp64/default.aspx)

**多态数据使用**

* 不要将指针强制转换成 int、long、ULONG 或 DWORD。

使用 UINT\_PTR、INT\_PTR、ULONG\_PTR 等。

例如：

[Copy Code](javascript:copycode('ctl00_maincontentcontainer_ctl06_code7');)

ImageBase = (PVOID) ((ULONG)ImageBase | 1);

应该改写成：

[Copy Code](javascript:copycode('ctl00_maincontentcontainer_ctl06_code6');)

ImageBase = (PVOID) ((ULONG\_PTR)ImageBase | 1);

* 使用 **PtrToUlong()** 和 **PtrToLong()** 截断指针：
  + 决不要将存储已截断指针的 int 或 ULONG 强制转换回指针。
  + 计算缓冲区大小时应该小心；大小可能超出 ULONG 的容量。
* 当调用具有指针 OUT 参数的函数时，应小心地使用正确的大小：

[Copy Code](javascript:copycode('ctl00_maincontentcontainer_ctl06_code5');)

void GetBufferAddress(OUT PULONG \*ptr);  
{  
\*ptr=0x1000100010001000;  
}  
void foo()  
{  
 ULONG bufAddress;  
 //  
 // this call causes memory corruption  
 //  
 GetBufferAddress((PULONG\*)&bufAddress);  
}

**对齐问题**

基于 Itanium 的系统需要自然对齐内存引用（也就是说，以 4 字节为边界访问 32 位）。错误对齐的内存引用在基于 Itanium 的系统上引发一个异常并对系统进行错误检测。

虽然 x64 更宽容，但要获取更好的性能，也需要对齐。

* 使用结构封装的指令时，应意识到对齐问题。
* 如果在单个头文件中使用不同的封装级别，应该格外小心。
* 为了获得最佳的结果，将 64 位值和指针放在结构的开始处。
* **RtlCopyMemory()** 和 **memcpy()** 不会发生错误。
* 使用 UNALIGNED 宏修复对齐问题：

[Copy Code](javascript:copycode('ctl00_maincontentcontainer_ctl06_code5');)

#pragma pack (1) /\* also set by /Zp switch \*/  
struct AlignSample {  
ULONG size;  
void \*ptr;   
};  
struct AlignSample s;  
void foo(void \*p) {  
\*p = p;// causes alignment fault  
...  
}  
foo((PVOID)&s.ptr);  
void foo(void \*p) {  
struct AlignSample s;   
 \*(UNALIGNED void \*)&s.ptr = p;   
}

有关对齐的更多信息，请参阅“ [内存管理：每个驱动程序作者都需要知道的技巧](http://www.microsoft.com/china/whdc/driver/tips/default.mspx)。”

**更多编码问题**

* 仔细地检查：
  + 含指针的显式和隐式联合体。
  + 存储在磁盘上或与 32 位进程进行交换的数据结构。
  + 安全描述符。
  + 处理内存区域大小的代码：

len = ptr2 - ptr1

len 可以大于 2^32

* 截断执行控制代码为 32 位。
* 使用 piecemeal-size 分配：

[Copy Code](javascript:copycode('ctl00_maincontentcontainer_ctl06_code5');)

struct foo {  
DWORD NumberOfPointers;  
PVOID Pointers[1];  
} xx;  
Wrong:  
malloc(sizeof(DWORD)+100\*sizeof(PVOID));   
Correct:  
malloc(FIELD\_OFFSET(struct foo,Pointers)  
+100\*sizeof(PVOID));

* 小心使用十六进制常量和无符号值。
* 当您使用无符号数作为下标时，应小心操作：
  + DWORD index = 0;
  + CHAR \*p;

表达式 **if** (p[index - 1] == '0') 将会在 Itanium 系统造成存取违规：

|  |  |
| --- | --- |
| **在 32 位机器上：** | **在 64 位机器上：** |
| p[index-1] == p[0xffffffff] == p[-1] | p[index-1] == p[0x00000000ffffffff] != p[-1] |

**其他注意事项**

* -1 != 0xFFFFFFFF

0xFFFFFFFF != 无效句柄

* DWORD 始终是 32 位。不要使用 DWORD 变量来存储指针或句柄。
* 将指针强制转换成 PCHAR 用于指针运算：

[Copy Code](javascript:copycode('ctl00_maincontentcontainer_ctl06_code2');)

ptr = (PVOID)((PCHAR)ptr + pageSize);

* 在调试语句中使用 %I 打印指针。
* 大于等于 0x80000000 的地址不一定是内核地址。

[返回页首](http://msdn.microsoft.com/zh-cn/windows/hardware/gg487358#top) [返回页首](http://msdn.microsoft.com/zh-cn/windows/hardware/gg487358#top)

**64 位 Windows 的驱动程序安装**

**64 位驱动程序 INF 需求**

Windows Server 2003 SP1 和更高的 Windows 版本不会在基于 x64 的系统上安装带有未修饰 INF 节的驱动程序包。

为了与 Intel Itanium 系统兼容，Windows Server 2003 SP1 将安装带有未修饰 INF 节的驱动程序包。但是，Windows 硬件徽标计划需要 INF 修饰，所以带有未修饰 INF 节的驱动程序包无法满足徽标的资格要求。

INF 的 **[Manufacturer]** 节条目和 **[Models]** 节名称必须针对将要安装在基于 x64 的系统上的驱动程序包添加修饰，而且应该针对基于 Intel Itanium 的系统添加修饰。原始发布版 Windows XP 之后的所有 Windows 版本都支持这些修饰；因此，通过这种方式修饰的非 x86 系统 INF 将适用于所有已发布的基于 NT 的 Windows 操作系统。

目标是防止安装错误的驱动程序二进制文件，因为客户不了解平台之间的差别。

* 使用 Windows XP 中引入的 TargetOsVersion。
* 在 Windows Server 2003 SP1 和以后的版本中，必须为 64 位平台修饰 [Manufacturer] 和 [Models] INF 节。例如：

[Copy Code](javascript:copycode('ctl00_maincontentcontainer_ctl06_code1');)

[Manufacturer] %mycompany% = MyCompanyModels, NTamd64 [MyCompanyModels.NTamd64] %MyDev% = mydevInstall, mydevHwid

**32 位和 64 位平台的驱动程序安装**

为了简化从 32 位安装程序安装 64 位驱动程序的过程，Microsoft 在 WDK 中提供 Driver Install Frameworks (DIFx) 工具的 64 位版本。DIFx 工具包括：

* Driver Package Installer (DPInst)。
* Driver Installation Frameworks for Applications (DIFxApp)。

无需为 64 位平台编写单独的 64 位安装程序，可以从单个 32 位安装程序启动 DPInst 或 DIFxApp 的正确版本。

无法使用 32 位驱动程序安装包直接安装 64 位驱动程序，但是可以使用它们确定当前正在运行哪个体系结构，然后启动合适的安装程序。在 64 位平台上，32 位安装程序在 WOW64 下运行，WOW64 就是让基于 Win32 的应用程序在 64 位 Windows 上运行的 x64 模拟器。安装程序必须能够检测其运行的平台，从而能够为该平台安装正确的驱动程序包。

[返回页首](http://msdn.microsoft.com/zh-cn/windows/hardware/gg487358#top) [返回页首](http://msdn.microsoft.com/zh-cn/windows/hardware/gg487358#top)

**开发 64 位驱动程序的参考资料**

* [WDK 中的“64 位问题”](http://msdn2.microsoft.com/en-us/library/aa489627.aspx)
* [IA-64 系统和 ACPI 2.0 64 位表格](http://www.microsoft.com/china/whdc/system/platform/64bit/ia64_acpi.mspx)
* [进一步了解 Windows 内存管理](http://msdn.microsoft.com/zh-cn/windows/hardware/gg463344)
* [超越 Windows XP：现在准备好迎接 64 位版本 Windows 的到来](http://msdn.microsoft.com/msdnmag/issues/01/11/xp64/default.aspx)
* [KMDF 驱动程序中的 DMA 支持](http://msdn.microsoft.com/zh-cn/windows/hardware/gg487384)
* [WHDC Web 站点上与硬件相关的信息](http://www.microsoft.com/china/whdc/system/platform/64bit/default.mspx)
* [64 位系统的 INF 需求](http://www.microsoft.com/china/whdc/driver/install/64inf_reqs.mspx)
* [基于 x64 的系统的修补策略](http://msdn.microsoft.com/zh-cn/windows/hardware/gg487350)
* [64 位 Windows 编程指南](http://msdn.microsoft.com/en-us/library/bb427430(vs.85).aspx)
* [内存管理：每个驱动程序作者都需要知道的技巧](http://msdn.microsoft.com/zh-cn/windows/hardware/gg487427)