Common Compiler Errors

This section illustrates the typical compiler errors that occur when migrating an existing code base. These examples happen to be from system-level HAL code, although the concepts are directly applicable to user-level code.

Warning C4311

'type cast' : pointer truncation from 'void *__ptr64 ' to 'unsigned long

Code

```
pPciAddr->u.AsULONG = (ULONG) CIA_PCI_CONFIG_BASE_QVA;
```

Description

PtrToUlong is an inline function or macro, depending on your usage. It truncates a pointer to a **ULONG**. Although 32-bit pointers are not affected, the upper half of a 64-bit pointer is truncated.

CIA_PCI_CONFIG_BASE_QVA is declared as a **PVOID**. The **ULONG** cast works in the 32-bit world, but results in an error in the 64-bit world. The solution is to get a 64-bit pointer to a **ULONG**, because changing the definition of the union that pPciAddr->u.AsULONG is defined in changes too much code.

Using the macro **PtrToUlong** to convert the 64-bit **PVOID** to the needed **ULONG** is allowed because we have knowledge about the specific value of CIA_PCI_CONFIG_BASE_QVA. In this case, this pointer will never have data in the upper 32 bits.

Solution

```
pPciAddr->u.AsULONG = PtrToUlong(CIA_PCI_CONFIG_BASE_QVA);
```

Warning C4311

'type cast' : pointer truncation from 'struct _ERROR_FRAME *__ptr64 ' to 'unsigned long

Code

```
KeBugCheckEx( DATA_BUS_ERROR,0,0,0,(ULONG)PUncorrectableError );
```

Description

The problem is that the last parameter to this function is a pointer to a data structure. Because PUncorrectableError is a pointer, it changes size with the programming model. The prototype for **KeBugCheckEx** was changed so that the last parameter is a **ULONG_PTR**. As a result, it's necessary to cast the function

pointer to a **ULONG_PTR**.

You might ask why **PVOID** was not used as the last parameter. Depending on the context of the call, the last parameter may be something other than a pointer or perhaps an error code.

Solution

```
KeBugCheckEx( DATA_BUS_ERROR,0,0,0,(ULONG_PTR)PUncorrectableError );
```

Warning C4244

'=' : conversion from 'struct _CONFIGURATION_COMPONENT *__ptr64 ' to 'struct _CONFIGURATION_COMPONENT *', possible loss of data

Code

```
Component = &CurrentEntry->ComponentEntry;
```

Description

The function declares the variable Component as a PCONFIGURATION_COMPONENT. Later, the variable is used in the following assignment that appears correct:

```
Component = &CurrentEntry->ComponentEntry;
```

However, the type PCONFIGURATION_COMPONENT is defined as:

```
typedef struct __CONFIGURATION_COMPONENT {
...
...
} CONFIGURATION_COMPONENT, * POINTER_32 PCONFIGURATION_COMPONENT;
```

The type definition for PCONFIGURATION_COMPONENT provides a 32-bit pointer in both 32-bit and 64-bit models because it is declared **POINTER_32**. The original designer of this structure knew it was going to be used in a 32-bit context in the BIOS and expressly defined it for that use. This code works fine in 32-bit Windows because the pointers happen to be 32-bit. In 64-bit Windows, it does not work because the code is in 64-bit context.

Solution

To work around this problem, use CONFIGURATION_COMPONENT * rather than the 32-bit PCONFIGURATION_COMPONENT . It is important to clearly understand the purpose of the code. If this code is intended to touch 32-bit BIOS or System space, this fix will not work.

POINTER_32 is defined in Ntdef.h and Winnt.h.

```
#ifdef (__AXP64__)
#define POINTER_32 _ptr32
#else
#define POINTER_32
#endif
```

Warning C4242

'=' : conversion from '__int64 ' to 'unsigned long ', possible loss of data

Code

```
ARC_STATUS HalpCopyNVRamBuffer (
IN PCHAR NvDestPtr,
IN PCHAR NvSrcPtr,
IN ULONG Length )
{

ULONG PageSelect1, ByteSelect1;

ByteSelect1 = (NvDestPtr - (PUCHAR)HalpCMOSRamBase) & CONFIG_RAM_BYTE_MASK;

ByteSelect1 = (NvDestPtr - (PUCHAR)HalpCMOSRamBase) & CONFIG_RAM_BYTE_MASK;
```

Description

This warning is generated because the calculation is using 64-bit values, in this case pointers, and placing the result in a 32-bit **ULONG**.

Solution

Type cast the result of the calculation to a **ULONG** as shown here:

```
ByteSelect1 = (ULONG)(NvDestPtr - (PUCHAR)HalpCMOSRamBase) & CONFIG_RAM_BYTE_MASK;
```

Typecasting the result lets the compiler know you are sure about the result. That being said, make certain you understand the calculation and really are sure it will fit in a 32-bit **ULONG**.

If the result may not fit in a 32-bit **ULONG**, change the base type of the variable that will hold the result.

Warning C4311

'type cast' : pointer truncation from 'void *__ptr64 ' to 'unsigned long '

Code

```
ULONG HalpMapDebugPort(
IN ULONG ComPort,
OUT PULONG ReadQva,
OUT PULONG WriteQva)
{
ULONG ComPortAddress;

ULONG PortQva;

// Compute the port address, based on the desired com port.

switch( ComPort ){
case 1:
   ComPortAddress = COM1_ISA_PORT_ADDRESS;
   break;

case 2:
default:
   ComPortAddress = COM2_ISA_PORT_ADDRESS;
}
PortQva = (ULONG)HAL_MAKE_QVA(CIA_PCI_SPARSE_IO_PHYSICAL) + ComPortAddress;

// Return the QVAs for read and write access.

*ReadQva = PortQva;
*WriteQva = PortQva;

return ComPortAddress;
}
```

Description

This entire function deals with addresses as integers, necessitating the need to type those integers in a portable way. All the local variables, intermediate values in calculations, and return values should be portable types.

Solution

```
ULONG_PTR HalpMapDebugPort(
IN ULONG ComPort,
OUT PULONG_PTR ReadQva,
OUT PULONG_PTR WriteQva)
{
ULONG_PTR ComPortAddress;

ULONG_PTR PortQva;

// Compute the port address, based on the desired com port.

switch( ComPort ){
case 1:
   ComPortAddress = COM1_ISA_PORT_ADDRESS;
   break;

case 2:
default:
   ComPortAddress = COM2_ISA_PORT_ADDRESS;
}

PortQva = (ULONG_PTR)HAL_MAKE_QVA(CIA_PCI_SPARSE_IO_PHYSICAL) + ComPortAddress;

// Return the QVAs for read and write access.

*ReadQva = PortQva;
*WriteQva = PortQva;

return ComPortAddress;
}
```

PULONG_PTR is a pointer that is itself 32 bits for 32-bit Windows and 64 bits for 64-bit Windows. It points to an unsigned integer, **ULONG_PTR**, that is 32 bits for 32-bit Windows and 64 bits for 64-bit Windows.

Warning C4311

'type cast' : pointer truncation from 'void *__ptr64 ' to 'unsigned long '

Code

```
BOOLEAN
HalpMapIoSpace (
VOID
)
{
PVOID PciIoSpaceBase;
PciIoSpaceBase = HAL_MAKE_QVA( CIA_PCI_SPARSE_IO_PHYSICAL );

//Map base addresses in QVA space.

HalpCMOSRamBase = (PVOID)((ULONG)PciIoSpaceBase + CMOS_ISA_PORT_ADDRESS);
```

Description

Even though all QVA (Quasi Virtual Address) values are really 32-bit values at this stage and will fit in a **ULONG**, it is more consistent to treat all addresses as **ULONG_PTR** values when possible.

The pointer PciIoSpaceBase holds the QVA that is created in the macro HAL_MAKE_QVA. This macro returns a 64-bit value with the top 32 bits set to zero so the math will work. We could simply leave the code to truncate the pointer into a **ULONG**, but this practice is discouraged to enhance code maintainability and portability. For example, the contents of a QVA might change in the future to use some of the upper bits at this level, breaking the code.

Solution

Be safe and use **ULONG_PTR** for all address and pointer math.

```
HalpCMOSRamBase = (PVOID)((ULONG_PTR)PciIoSpaceBase + CMOS_ISA_PORT_ADDRESS);
```

Warning C4311

'type cast' : pointer truncation from 'void *__ptr64 ' to 'unsigned long '

Code

```
PVOID
HalDereferenceQva(
PVOID Qva,
```

```
INTERFACE_TYPE InterfaceType,
ULONG BusNumber)

if ( ((ULONG) Qva & QVA_SELECTORS) == QVA_ENABLE ) {

return( (PVOID)( (ULONG)Qva << IO_BIT_SHIFT ) );
}
else {
return (Qva);
}
```

Description

The compiler warns about the address of (&) and left shift (<<) operators if they are applied to pointer types. In the above code, Qva is a **PVOID** value. We need to cast that to an integer type to perform the math. Because the code must be portable, use **ULONG_PTR** instead of **ULONG**.

Solution

```
if ( ((ULONG_PTR) Qva & QVA_SELECTORS) == QVA_ENABLE ) {
  return( (PVOID)( (ULONG_PTR)Qva << IO_BIT_SHIFT ) );
```

Warning C4311

'type cast' : pointer truncation from 'void *__ptr64 ' to 'unsigned long '

Code

```
TranslatedAddress->LowPart = (ULONG)HalCreateQva(
*TranslatedAddress, va);
```

Description

TranslatedAddress is a union that looks something like the following:

```
typedef union
   Struct {
```

```
        ULONG LowPart;
        LONG Highpart;
    }
    LONGLONG QuadPart;
}
```

Solution

Knowing what the rest of the code might place in Highpart, we can select either of the solutions shown here.

```
TranslatedAddress->LowPart = PtrToUlong(HalCreateQva(*TranslatedAddress,va) );
```

The **PtrToUlong** macro truncates the pointer returned by **HalCreateQva** to 32 bits. We know that the QVA returned by **HalCreateQva** has the upper 32 bits set to zero and the very next line of code sets TranslatedAddress->Highpart to zero.

With caution, we could use the following:

```
TranslatedAddress->QuadPart = (LONGLONG)HalCreateQva(*TranslatedAddress,va);
```

This works in this example: the **HalCreateQva** macro is returning 64 bits, with the upper 32 bits set to zero. Just be careful not to leave the upper 32 bits undefined in a 32-bit environment, which this second solution may actually do.

Warning C4311

'type cast' : pointer truncation from 'void *__ptr64 ' to 'unsigned long '

Code

```
VOID
HalpCiaProgramDmaWindow(
PWINDOW_CONTROL_REGISTERS WindowRegisters,
PVOID MapRegisterBase
)
{
CIA_WBASE Wbase;

Wbase.all = 0;
Wbase.Wen = 1;
Wbase.SgEn = 1;
Wbase.Wbase = (ULONG)(WindowRegisters->WindowBase) >> 20;
```

Description

WindowRegisters->WindowBase is a pointer and is now 64 bits. The code says to right-shift this value 20 bits. The compiler will not let us use the right-shift (>>) operator on a pointer; therefore, we need to cast it to some sort of integer.

Solution

```
Wbase.Wbase= PtrToUlong ( (PVOID) ((ULONG_PTR) (WindowRegisters->WindowBase) >> 20));
```

Casting to a **ULONG_PTR** is just what we need. The next problem is Wbase. Wbase is a **ULONG** and is 32 bits. In this case, we know that the 64-bit pointer WindowRegisters->WindowBase is valid in the lower 32 bits even after being shifted. This makes use of the **PtrToUlong** macro acceptable, because it will truncate the 64-bit pointer into a 32-bit **ULONG**. The **PVOID** cast is necessary because **PtrToUlong** expects a pointer argument. When you look at the resulting assembler code, all this C code casting becomes just a load quad, shift right, and store long.