

JD Edwards EnterpriseOne 64-bit Compiler Warnings and C Programming Recommendations

ORACLE BUSINESS BRIEF / JUNE 30, 2020

ORACLE®

DISCLAIMER

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Table of Contents

INTRODUCTION	5
POINTER.....	6
Improper Use of Pointer	6
Typecasting of Pointers	7
Pointer Math	8
Integral Variables.....	10
Time_t	10
Integer Truncation	11
Integral Math and Integer Overflow	12
Integral Promotion Changing Sign and Width.....	14
64-bit Functional	15
Decorated Function Names.....	15
Deprecation	16
Compiler Restrictions.....	18
Function Prototyping.....	18
Appendix A – Backward Compatibility	19
Data Types	19
APIs.....	19

MATH_NUMERIC APIs	20
Arithmetic APIs	20
Memory APIs	20
Macro Statements	21
Appendix B - 64-bit Native Data Types by Platform	29
Appendix C – Compiler Warnings	30

INTRODUCTION

This document contains concepts and coding suggestions pertaining to the building of business function C code for 64-bit EnterpriseOne server and client platforms. Some of the resolutions presented here utilize data types and APIs that are only available in Tools Release 9.2.2 and later. To accommodate backward compatibility with earlier Tools releases, refer to Appendix A.

Each concept and coding suggestion is presented in the following format:

Description

A short description of the concept or code error you might experience while building 64-bit business functions.

Message Text

Examples of possible compiler errors or warning messages.

Example Code

An example of uncompliant code.

Discussion

A short description of why the coding is not compliant on a 64-bit platform.

Recommendation

A Short description of how to correct the code in the message text.

Recommendation Code

An example of compliant 64-bit coding for the current example.

Platform

Unique platform-specific differences.

POINTER

The three common situations that arise with pointers when compiling your source code in 64-bit are:

- » Improper use of pointers
- » Typcasting of pointers
- » Point math

Improper Use of Pointers

Assigning pointers to integral variables and vice versa will generate warnings in the 64-bit compiles.

You use `jdeStoreDataPtr`, `jdeRetrieveDataPtr`, and `jdeRemoveDataPtr` APIs to pass pointers to and from functions. Passing pointers incorrectly or assigning pointers to integral variables will lead to errors.

MESSAGE TEXT

- » warning C4133: 'function' : incompatible types - from 'int **' to 'LONG64 **'
- » warning C4305: 'type cast' : truncation from 'HCACHE' to 'unsigned int'
- » warning C4305: 'type cast' : truncation from 'HREQUEST' to 'ID'
- » warning C4305: 'type cast' : truncation from 'LPVOID' to 'int'
- » warning C4305: 'type cast' : truncation from 'NID' to 'ID'
- » warning C4305: 'type cast' : truncation from 'void **' to 'int'
- » warning C4305: 'type cast' : truncation from 'void **' to 'JDEDB_LEVEL'
- » warning C4306: 'type cast' : conversion from 'ID' to 'HREQUEST' of greater size
- » warning C4306: 'type cast' : conversion from 'ID' to 'JCHAR **' of greater size
- » warning C4306: 'type cast' : conversion from 'ID' to 'void **' of greater size
- » warning C4306: 'type cast' : conversion from 'int' to 'int **' of greater size
- » warning C4306: 'type cast' : conversion from 'int' to 'LPVOID' of greater size
- » warning C4306: 'type cast' : conversion from 'int' to 'void **' of greater size

EXAMPLE CODE

```
lpDS->idFileHandle = (unsigned long) lpF01133THdl;  
lpDS->idhCursor = (ID) jdeRemoveDataPtr(hUser, (unsigned long) lpDS->idhCursor);
```

DISCUSSION

Use the following Data Pointer Storage APIs instead of casting a pointer to an unsigned long to be stored in an ID parameter:

- » `jdeRemoveDataPtr`
- » `jdeRetrieveDataPtr`
- » `jdeStoreDataPtr`

RECOMMENDATION

The recommendation for business function source code is to use the Data Pointer APIs that provide the tooling for passing pointer values between functions.

RECOMMENDATION CODE

```
lpDS->idFileHandle = jdeStoreDataPtr(hUser, lpF01133THdl);  
HJDECURSOR hCursor = jdeRemoveDataPtr(hUser, lpDS->idhCursor);
```

PLATFORM

NA

Typecasting of Pointers

Typecasting values may hide problems that exist in your current code. If the compiler is issuing type mismatch warnings, do not add a cast to work with the compiler.

MESSAGE TEXT

- » Warning C4047: 'function' : 'PJSTR' differs in levels of indirection from 'ID'
- » Warning C4024: 'jdeStrcpy' : different types for formal and actual parameter 1
- » Warning C4074: 'function' : 'LPINT' differs in levels of indirection from 'int'
- » Warning C4024: 'MathNumericToInt' : different types for formal and actual parameter 2

EXAMPLE CODE

```
ID                idErrorMessageID;

jdeStrcpy((JCHAR *)idErrorMessageID, _J("078N"));

int                nMaxCatalogLevel    = 0;

MathNumericToInt(&lpDS->nMaxCatalogTreeLevel, (int *)nMaxCatalogLevel);
```

DISCUSSION

In the example above, adding the (JCHAR *) and (int*) typecast makes the compiler work but does not fix the issue. Copying the string (_J("078N")) above into an integral variable will not resolve the issue.

RECOMMENDATION

You want to minimize the use of typecasting. There are occasions when typecasting is necessary and appropriate, but in most cases typecasting is not necessary.

In the example above, you should be copying the string into a string variable, and the MathNumericToInt API needs a pointer to an int for the second parameter, not the value itself.

RECOMMENDATION CODE

```
JCHAR                szErrorMessageID[11];
jdeStrcpy(szErrorMessageID, _J("078N"));
code

MathNumericToInt(&lpDS->nMaxCatalogTreeLevel, &nMaxCatalogLevel);
```

PLATFORM

NA

Pointer Math

It is common to use pointer math to generate the count of objects in a block of memory. This difference will result in a 64 b width value. For example:

- » $p2 - p1$ (Gives the offset of $p2$ from $p1$.)
- » $p2 - p1 + 1$ (Might give the number of objects in an object list or an array.)

MESSAGE TEXT

- » AIX64 - 1506-742 (I) 64-bit portability: possible loss of digits through conversion of long int type into int type.
- » HP64 - warning #4229-D: 64-bit migration: conversion from "long" to "int" may truncate value
- » WIN64 - warning C4242: '=' : conversion from '_int64' to 'int', possible loss of data

EXAMPLE CODE

```
JCHAR *p1, *p2;

    long lStrSize=0;

    lStrSize = (long) (p2 - p1);
```

DISCUSSION

In this code, you have the difference of two pointers. The WIN64 warning message states from "...from '_int64' ...". This is how WIN64 internally represents the rvalue of $(p2 - p1)$. On WIN64, the difference of two pointers is a signed 64-bit value. Similarly, on AIX and HP, the rvalue is a 'long', a 64-bit signed int.

```
lStrSize = (long) (p2 - p1);
```

With respect to `lStrSize`, on some platforms, a long is a 64-bit integer, whereas on others such as WIN64, it is not. On WIN64, a long is a 32-bit integer, so the difference of 2 64-bit pointers will be truncated when it is assigned to the long variable `lvalue`.

The warning indicates "... to 'int' ..." while the variable `lStrSize` is defined as long. In the code conversion from 32-bit to 64-bit, all long data types will be changed from long to int. The message above is what you will see when 64-bit compiles even though the variable is currently defined as long.

RECOMMENDATION

There are two possible ways to approach this situation:

1. Use `jdeCast32()` to downcast the int64 value to a 32-bit int.

```
lStrSize = jdeCast32 (p2 - p1);
```

This is a quick approach and requires minimal modification of existing code. However, it will result in a manageable truncation of a 64-bit value. It may not be a valid recommendation in certain application usages where the value might be less than -2 GB or greater than 2 GB (greater than 4 GB in the case of `jdeCast32U()`).

It adds an additional invocation stack entry, utilizing more system resources.

2. Change the data type of `lStrSize` to `jdesizeV_t` and use 64-bit `size_t` versions of string APIs. This recommendation eliminates truncation; however, the code modifications are more involved and customized to the specific situation:

```
jdesizeV_t lStrSize = 0;

if (p2)
{
```

```

        lStrSize = p2 - p1;
    }
    Else
    {
        lStrSize = jdeStrlenV(p1);
    }
    jdeStrncpyV(szRetValue, p1, lStrSize);
    szRetValue[lStrSize] = 0;

```

The difference of pointers means that the lStrSize needs to be a 64-bit data type in 64-bit mode. The lStrSize is later used in size_t related function calls such as jdeStrlen and jdeStrncpy:

» jdeStrlen returns a size_t value.

» jdeStrncpy needs a size_t value in the third parameter.

These suggest that a jdesizeV_t (64-bit size_t on every platform) should be used instead of *long* or *int*.

Other options might be:

» jdesizeV_t is a 64-bit signed integer.

» ptrdiff_t is the native difference between 2 pointers. This is a 64-bit value on all platforms.

RECOMMENDATION CODE

```

1. lStrSize = jdeCast32(p2 - p1);

2. jdesizeV_t lStrSize = 0;
   if (p2)
   {
       lStrSize = p2 - p1;
   }
   Else
   {
       lStrSize = jdeStrlenV(p1);
   }
   jdeStrncpyV(szRetValue, p1, lStrSize);
   szRetValue[lStrSize] = 0;

```

PLATFORM

See the Message Text section above.

INTEGRAL VARIABLES

Integrals are any integer class data type value. For example, int, long, unsigned int, size_t, time_t are all integral data types. In 64-bit platforms, long, size_t, time_t data types can change to be 64-bit width. Using these variables results in a truncation of values.

time_t

The time_t data type is an ISO C standard data type. You use this data type to indicate the time in seconds where the value 0 is Thu Jan 01 00:00:00 1970. The largest time_t value in 32-bit platforms is 2038-01-19 03:14:07 UTC, which is a known technical C language limitation and is resolved in 64-bit platforms.

MESSAGE TEXT

WIN64 warning C4242: '=' : conversion from 'time_t' data to 'int', possible loss of data

WIN64 warning C4133: 'function' : incompatible types - from 'const int *' to 'const time_t *'

EXAMPLE CODE

```
time_t  lTime;

Short nHours, nMinutes, nSeconds;

lTime = nHours * 10000 + nMinutes * 100 + nSeconds;

LongToMathNumeric((long)lTime, &lpDS->mnTimeLastUpdated );

int      lTime;

lTime = time(NULL);

jdeLocaltime(&lTime, &dsLocalTime);
```

DISCUSSION

Often, time_t variables are used as general integer variables, such as int or long as general purpose variables, and not as elapsed seconds since 1970. In the first example above, lTime is used to maintain an HHMMSS, which is not elapsed seconds. The variable is passed to a function that does not take time_t. In the second example, int and long variables are incorrectly used, where time_t would be appropriate.

RECOMMENDATION

Use time_t for variables in the following functions only:

- » time ()
- » currenttime()
- » gmtime()
- » localtime()

Use int variables to maintain HHMMSS values.

RECOMMENDATION IN CODE

```
long  lTime;

Short nHours, nMinutes, nSeconds;

lTime = nHours * 10000 + nMinutes * 100 + nSeconds;

LongToMathNumeric(lTime, &lpDS->mnTimeLastUpdated );
```

```
time_t      lTime;
tTime = time (NULL);
jdeLocaltime(&lTime, &dsLocalTime);
```

PLATFORM

NA

Integer Truncation

Integer truncation occurs in an expression when the source (R-value) is too large for the target variable or function parameter (L-value).

MESSAGE TEXT

- » AIX64 - Warning #1506-742 (I) 64-bit portability: possible loss of digits through conversion of long int type into unsigned int type.
- » HP64 - Warning #4229-D: 64-bit migration: conversion from "long" to "int" may truncate value.
- » HP64 - Warning #4229-D: 64-bit migration: conversion from "long" to "jdesize_t" may truncate value.
- » HP64 - Warning #4229-D: 64-bit migration: conversion from "off_t" to "int" may truncate value.
- » WCLT64 - Warning C4242: '=' : conversion from 'intptr_t' to 'int', possible loss of data.

EXAMPLE CODE

```
struct jdeZStat  stbuf; //stbuf.st_size member is defined as off_t type
int             filesize = 0;
jdeStat(filePath, &stbuf);
...
filesize = stbuf.st_size;
-----^
```

DISCUSSION

In the example above, the structure member `stbuf.st_size` will be an `off_t` (64-bit signed value). The target variable `filesize` is too small to contain the complete value of `stbuf.st_size`. The high-order-bits of the `stbuf.st_size` value will be cut off and thrown away as it is copied to the `filesize` variable.

RECOMMENDATION

You need to make the *lvalue* and *rvalues* the same width in the expression in order to eliminate the warning. Possible resolutions are to:

1. Make the target variable larger.
2. Make the source value smaller. That is not possible in this example because the `jdeZStat` structure is defined by the compiler.
3. Use the new `jdeCastxx` family of functions. These functions are for JDE Business functions where a “down-cast” is required.
 - a. `jdeCast32` | `jdeCast32U`

These functions will accept a 64-bit value, return the truncated 32-bit value (signed or unsigned), and if significant digits are lost, a message is written to the jde.log.

b. `jdeCast16` | `jdeCast16U`

These functions will accept a 64-bit value, return the truncated 16-bit value (signed or unsigned), and finally, if significant digits are lost, a message is written to the jde.log.

c. `jdeTrunc32` | `jdeTrunc32U`

These functions will accept a 64-bit value, return the truncated 32-bit value (signed or unsigned) with no log entry.

RECOMMENDATION CODE

```
struct jdeZStat  stbuf;

off_t           filesize1 = 0;
jdessizeV_t     filesize2 = 0;
int             filesize3 = 0;

jdeStat(filePath, &stbuf);
filesize1 = stbuf.st_size;
filesize2= stbuf.st_size;
filesize2= jdeCast32(stbuf.st_size);
```

PLATFORM

NA

Integral Math and Integer Overflow

An integer overflow occurs when an arithmetic operation attempts to create a numeric value that exceeds the range represented by the number of bits. The value is either larger than the maximum or lower than the minimum representable value.

MESSAGE TEXT

- » warning #4298-D: addition result could be truncated before cast to bigger sized type
- » warning #4298-D: subtraction result could be truncated before cast to bigger sized type
- » warning #4299-D: multiply result could be truncated before cast to bigger sized type

EXAMPLE CODE

```
unsigned int     nLen = 0;

...

len = lpWOBlob->lSize;

jdeToUnicode(szXMLCF, (ZCHAR*) lpWOBlob->lpValue, nLen+1, _J("cp1252"));

-----^
```

DISCUSSION

The `jdeToUnicode` function expects a `jde_n_charV` value for the number of bytes for the function call. `jde_n_charV_t` (aka `size_t`) is an unsigned 64-bit integer in 64-bit platforms. The `unsigned int` variable `nLen`. will be 32-bits in all cases. You have a 32-bit value being assigned to a 64-bit function variable. The compiler will be making adjustments when the `nLen` value is passed to the `jdeToUnicode` function.

The `nLen` expression by itself is a simple integral promotion example; there is no conflict and no warning. When the "+ 1" gets included in the expression, there are other factors to be considered. For example, if the value of `nLen` is `INT_MAX (0xFFFFFFFF)` then, `nLen` would overflow and the resulting value would become 0 (`0x00000000`). You have effectively changed the value that is being passed.

Overflow is an issue that must be considered in every C expression. Compilers seldom issue warnings for expressions that could overflow, particularly when the R-value and L-value of the expression have the same width.

In the example above, the R-value and L-value of the expression are not the same width. However, there is a way to prevent the overflow concern. On HPUX, the compiler recognizes this and provides a warning. By up-casting the `nLen` value prior to the "+ 1" addition, we define an R-value that can accommodate the Integer math without overflowing.

RECOMMENDATION

You need to make the `lvalue` and `rvalues` the same width in the expression in order to eliminate the warning.

The following are possible resolutions:

- Adjust the value prior to making the function call. The calculation is applied to the 32-bit L-value variable.
- Change the variable to match the data type expected by `jdeToUnicode()` . The R-Value of the `nLen +1` expression then becomes 64-bit width.
- Up-cast `nLen` before adding 1. The R-Value of the `nLen` in the expression then becomes 64-bit width and 1 is then added to the 64-bit.

RECOMMENDATION CODE

```
nLen = lpWOBlob->lSize + 1;
jdeToUnicode(szxMLCF, (ZCHAR*) lpWOBlob->lpValue, nLen, _J("cp1252"));

jdeSizeV_t      nLen = 0;    //
nLen = lpWOBlob->lSize;
jdeToUnicode(szxMLCF, (ZCHAR*) lpWOBlob->lpValue, nLen+1, _J("cp1252"));

jdeToUnicode(szxMLCF, (ZCHAR*) lpWOBlob->lpValue, (size_t)nLen+1,
_J("cp1252"));
```

PLATFORM

These warnings only occur on the HPUX 64-bit platform.

Integral Promotion Changing Sign and Width

When you change the sign and width of a value in an expression, warnings may occur. This type of expression can result in undefined behavior based on the rules of Integral Promotion.

MESSAGE TEXT

1506-743 (I) 64-bit portability: possible change of result through conversion of int type into unsigned long int type

1506-743 (I) 64-bit portability: possible change of result through conversion of short type into unsigned long int type

1506-743 (I) 64-bit portability: possible change of result through conversion of unsigned int type into long int type

EXAMPLE CODE

```
short nSizeOfDataStructure;
...
memset((void *) GenericLpDSKey, (int) _J('\0'), nSizeOfDataStructure);
```

DISCUSSION

The *memset* function expects a *size_t* value for the number of bytes for the function call. *size_t* is an unsigned 64-bit integer in 64-bit platforms. A *short* defines a 16-bit signed value. When you use a short value for the third parameter of *memset*, the compiler must change both the sign and width of the *nSizeOfDataStructure* value, which violates the rules of Integral Promotion. In the example above, the variable is defined with the wrong data type *short*. The size of a data structure can never be negative. Therefore, the proper choice for variable type is *size_t* or at least an unsigned integral.

RECOMMENDATION

Use the correct data type value expected by a function call. In the example above, the proper data type to use is *size_t*, *memset* expects a value of type *size_t*.

RECOMMENDATION CODE

```
size_t nSizeOfDataStructure;
memset((void *) GenericLpDSKey, (int) _J('\0'), nSizeOfDataStructure);
or
unsigned short nSizeOfDataStructure;
memset((void *) GenericLpDSKey, (int) _J('\0'), nSizeOfDataStructure);
```

PLATFORM

These errors only occur on AIX 64-bit platforms.

64-BIT FUNCTIONAL

New warnings are introduced in 64-bit platforms due to the changes and improvements within the compilers themselves. The new warnings are 64-bit specific because they appear in the 64-bit platforms, but not in the 32-bit platforms.

Decorated Function Names

Decorated function names are required in a Windows 32-bit environment; Windows 64-bit builds use simple function names defined in the source.

MESSAGE TEXT

No compiler warning message.

EXAMPLE CODE

```
myProc = GetProcAddress( hLib, "_GetServicePackReleaseAPI@4" );
--or--

#ifdef JDENV_PC
    jdeStrcpy(szXMLAPI, _J("_xmlReq_convertFromUTF8ToUnicode@12"));
#else
    jdeStrcpy(szXMLAPI, _J("xmlReq_convertFromUTF8ToUnicode"));
#endif
```

DISCUSSION

Dynamic Windows 32-bit function calls require a decorated function name. In the example above, the actual function name is "xmlReq_convertFromUTF8ToUnicode." However, the Windows 32-bit function name string is "_xmlReq_convertFromUTF8ToUnicode@12". 32-bit Windows requires a leading "_" and a trailing "@###" where ### is the sum of bytes used for the function parameters. 64-bit Windows does not require this decoration and uses just the actual function name. 64-bit Windows now behaves the same as the Unix and IBM i platforms.

RECOMMENDATION

Change the Windows-only clause to also exclude:

```
#if defined(JDENV_PC) && !defined(JDENV_64BIT)
    ...Windows 32-bit decorated function string here
#else
    ... Undecorated function string here
#endif
```

RECOMMENDATION CODE

```
#if defined(JDENV_PC) && !defined(JDENV_64BIT)
    myProc = GetProcAddress(hLib, "_GetServicePackReleaseAPI@4");
#else
    myProc = GetProcAddress(hLib, "GetServicePackReleaseAPI");
```

```

#endif

--or--

#if defined(JDENV_PC) && !defined(JDENV_64BIT)
    jdeStrcpy(szXMLAPI, _J("_xmlReq_convertFromUTF8ToUnicode@12"));
#else
    jdeStrcpy(szXMLAPI, _J("xmlReq_convertFromUTF8ToUnicode"));
#endif

```

PLATFORM

Decorated Function Names occur only on Windows platforms, but the change must be made for all platforms.

Deprecation

There are two existing JD Edwards code statements that are no longer required. These two code statements generate a warning on 64-bit platform builds.

A warning is generated on all build platforms.

error #2020: identifier "GPoolCommon" is undefined

warning C4305: 'type cast' : truncation from 'HWND' to 'GLRTID'

EXAMPLE CODE

The first deprecated line(s):

```

GPoolCommon = jdeMemoryManagementInit();
GPoolCommon = NULL;

```

The first deprecated line(s):

```

dsUbeStructure.idRunTime = (GLRTID)lpdsInternal->lpBhvrCom->hDlg << 16;

```

DISCUSSION

The GPoolCommon feature has been deprecated. In 64-bit builds, the global variable is no longer available. It is only maintained in 32-bit builds for backward compatibility.

The idRunTime variable of the UBE Call structure is no longer used.

RECOMMENDATION

You may comment out or delete these source code lines.

RECOMMENDATION CODE

```

/*    GPoolCommon = jdeMemoryManagementInit();    */
// dsUbeStructure.idRunTime = (GLRTID)lpdsInternal->lpBhvrCom->hDlg << 16;

```

PLATFORM

NA

COMPILER RESTRICTIONS

Compilers have continued to evolve and continue to introduce compliance restrictions.

Function Prototyping

IBM i C compilers:

IBM i C compiler issues a warning if a function prototype contains no parameters. The standard is that the use of function declarators with empty parentheses (not prototype-format parameter type declarators) is an obsolescent feature.

MESSAGE TEXT

CZM0304 No function prototype given for "ApiUnit_Density_API".

EXAMPLE CODE

Prototype definition: `ApiUnit * ApiUnit_Density_API();`

Function definition: `ApiUnit * ApiUnit_Density_API() {...}`

DISCUSSION

The IBM i C compiler follows the C99 C standard, where every function must have a defined list of parameters. If no parameters are required, then the parameter list is void. In this specific warning, the compiler does not like that the function is defined without a specific variable definition. The warning is issued where the function is used, so the correction must be made to the function where it is defined.

RECOMMENDATION

The warning is issued where the function is used, so the correction must be made to the function where it is defined.

You will want to verify that the header that defines the prototype is included in the source.

RECOMMENDATION CODE

Prototype definition: `ApiUnit * ApiUnit_Density_API(void);`

Function definition: `ApiUnit * ApiUnit_Density_API(void) {...}`

PLATFORM

These errors only occur on IBM i platforms.

APPENDIX A – BACKWARD COMPATIBILITY

The solutions offered in this document may include JD Edwards EnterpriseOne data types, API functions, or macros that are only available in Tools Release 9.2.2 and later. However, this does not prevent you from using the solution in Applications 9.2 and Applications 9.1 with earlier Tools releases. JD Edwards EnterpriseOne has provided backward compatible options that allow you to update your Applications 9.1 and Applications 9.2 business function source. You only need to add a simple definition to the object source to make the code change backward compatible to previous Tools releases. The new macros will map the Tools Release 9.2.2 API name to the complementary API already defined in previous Tools releases.

The Pointer Math recommendations include the use of the new functions. These APIs are fully defined in Tools Release 9.2.2, but are not available in prior Tools releases. For example, new functions are illustrated in bold font below:

```
» IStrSize = jdeCast32(p2 - p1);
» jdesizeV_t IStrSize = 0;
   IStrSize =jdeStrlenV(p1);
   jdeStrncpyV(szRetVal, p1, IStrSize);
```

Most of the JDEUNICODE.H APIs that exist for 32-bit implementations have complementary 64-bit capable APIs. The naming convention for these standard string APIs is:

32-bit Data type	jdesize_t	64-bit Data type	jdesizeV_t
32-bit Function Name	jdeStrlen()	64-bit Function Name	jdeStrlenV()
32-bit macro	DIM()	64-bit macro	jdeStrlenV()

NOTE: THE V IN THE NAME INDICATES THAT THIS FUNCTION USES OR RETURNS A VALUE WITH A WIDTH THAT VARIES DEPENDING ON THE THE 32-BIT OR 64-BIT PLATFORM DEFINITION, FOR EXAMPLE, VALUES ARE 32-BIT IN 32-BIT PLATFORMS AND 64-BIT IN 64-BIT PLATFORMS.

When an API is recommended in a resolution and it does not exist in your Tools release, use the macro statements defined in Macro Statements section below for the API in question. Simply add the macro definition for the required API to the bottom of the Bxxxxxxx.h header file in the object source code where the API is used.

Data Types

Data Type	Return Value Width in 64-bit builds	Alternative Solution	64-bit Width
size_t jdesize_t	32-bit unsigned value	jdesizeV_t *	64-bit unsigned value
ssize_t jdessize_t	32-bit signed value	jdessizeV_t *	64-bit signed value
jde_n_byte	32-bit unsigned value	jde_n_byteV *	64-bit unsigned value
jde_n_char	32-bit unsigned value	jde_n_charV *	64-bit unsigned value

APIs

32-bit Function	Return Value Width in 64-bit builds	Alternative Solution	64-bit Width
DIM()	32-bit unsigned value	DIMV()	64-bit unsigned value
DIM_NID()	32-bit unsigned value	DIM_NIDV()	64-bit unsigned value

32-bit Function	Return Value Width in 64-bit builds	Alternative Solution	64-bit Width
Sizeof jdesizeof	32-bit unsigned value	jdesizeofV	64-bit unsigned value
jdeStrlen()	32-bit unsigned value	jdeStrlenV()	64-bit unsigned value
jdeStrncpy()	32-bit unsigned value	jdeStrncpyV()	64-bit unsigned value
jdeStrnicmp()	32-bit unsigned value	jdeStrnicmpV()	64-bit unsigned value
jdeStrncmp()	32-bit unsigned value	jdeStrncmpV()	64-bit unsigned value
jdeStrncmpPosix()	32-bit unsigned value	jdeStrncmpPosixV()	64-bit unsigned value
jdeStrnicmpPosix()	32-bit unsigned value	jdeStrnicmpPosixV()	64-bit unsigned value
jdeStrncat()	32-bit unsigned value	jdeStrncatV()	64-bit unsigned value
jdeStrftime()	32-bit unsigned value	jdeStrftimeV()	64-bit unsigned value
jdeSnprintf()	32-bit unsigned value	jdeSnprintfV()	64-bit unsigned value
jdeZStrlen()	32-bit unsigned value	jdeZStrlenV()	64-bit unsigned value
jdeZStrnicmp()	32-bit unsigned value	jdeZStrnicmpV()	64-bit unsigned value
jdeVsnprintf()	32-bit unsigned value	jdeVsnprintfV()	64-bit unsigned value
jdeZStrlenSigSafe()	32-bit unsigned value	jdeZStrlenSigSafeV()	64-bit unsigned value

MATH_NUMERIC APIs

32-bit Function	Return Value Width in 64-bit builds	Alternative Solution	64-bit Width
TimeToMathNumeric()	64-bit width time_t	TimeToMathNumeric()	64-bit width time_t value

Arithmetic APIs

Function	64-bit Build Width
jdeCast16()	return a 16-bit signed value
jdeCast16U()	return a 16-bit unsigned value
jdeCast32()	return a 32-bit signed value
jdeCast32U()	return a 32-bit unsigned value

Memory APIs

Current 32-bit Function	64-bit Build Width	Alternative Solution	64-bit Width
memset() jdeZMemset()	32-bit unsigned value	jdeZMemsetV()	64-bit unsigned value
memcpy() jdeMemcpy()	32-bit unsigned value	jdeMemcpyV()	64-bit unsigned value
memcmp() jdeMemcmp()	32-bit unsigned value	jdeMemcmpV()	64-bit unsigned value
memmove() jdeMemmove()	32-bit unsigned value	jdeMemmoveV()	64-bit unsigned value
jdeAlloc()	32-bit unsigned value	jdeAllocV()	64-bit unsigned value
jdeAllocReturnIfError()	32-bit unsigned value	jdeAllocReturnIfErrorV()	64-bit unsigned value
jdeReAllocReturnIfError()	32-bit unsigned value	jdeReAllocReturnIfErrorV()	64-bit unsigned value

Current 32-bit Function	64-bit Build Width	Alternative Solution	64-bit Width
jderealloc()	32-bit unsigned value	jdereallocV()	64-bit unsigned value
malloc() jdemalloc()	32-bit unsigned value	jdemallocV()	64-bit unsigned value
realloc() jdeReAllocEx()	32-bit unsigned value	jdeReAllocExV()	64-bit unsigned value
calloc() jdecalloc()	32-bit unsigned value	jdecallocV()	64-bit unsigned value
fread() jdeFread()	32-bit unsigned value	jdeFreadV()	64-bit unsigned value
fwrite() jdeFwrite()	32-bit unsigned value	jdeFwriteV()	64-bit unsigned value

Macro Statements

size_t

The size_t data type will change between 32-bit and 64-bit on Unix platforms. However, it remains 32-bit on IBM i. The jdesizeV_t and jdessizeV_t will provide a standard data type with a width that varies but is consistent on all platforms.

```
#if !defined(JDENV_64BIT) && !defined(jdesizeV_t)
#   define jdesizeV_t          unsigned int
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jdessizeV_t)
#   define jdessizeV_t        int
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jdesizeofV)
#   define jdesizeofV         sizeof
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jde_n_byteV)
#   define jde_n_byteV        jde_n_byte
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jde_n_charV)
#   define jde_n_charV        jde_n_char
#endif
```

Some Business Function code (BSFNs) may need to make calls to third-party API's, for example; Operating System (OS) or other non-EnterpriseOne software. If these API's take as parameters or return long or unsigned long datatypes, then the BSFN should continue to pass and receive these types. In order to use the long and unsigned long types in a cross-platform manner, JD Edwards types have been defined and should be used in BSFNs where applicable. Using the following JD Edwards types will prevent them from getting converted when the source code is converted from 32- to 64-bit:

```
#if !defined(JDENV_64BIT) && !defined(jdelong)
#   define jdelong long
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jdeulong)
#   define jdeulong unsigned long
#endif
```

time_t

The time_t datatype will change between 32-bit and 64-bit. The jdetime_t will standardize these differences.

```
#if !defined(JDENV_64BIT) && !defined(jdetime_t)
#   define jdetime_t          time_t
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(DIMV)
#   define DIMV              DIM
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(DIM_NIDV)
#   define DIM_NIDV         DIM_NID
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jdeFreadV)
#   define jdeFreadV        jdeFread
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jdeFwriteV)
#   define jdeFwriteV       jdeFwrite
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jdeMemcmpV)
```

```
# define jdeMemcmpV          memcmp
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jdeMemcpyV)
# define jdeMemcpyV          memcpy
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jdeMemmoveV)
# define jdeMemmoveV          memmove
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jdeZMemsetV)
# define jdeZMemsetV          memset
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jdeSnprintfV)
# define jdeSnprintfV          jdeSnprintf
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jdeStrftimeV)
# define jdeStrftimeV          jdeStrftime
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jdeStrlenV)
# define jdeStrlenV            jdeStrlen
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jdeStrncatV)
# define jdeStrncatV           jdeStrncat
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jdeStrncmpPosixV)
# define jdeStrncmpPosixV      jdeStrncmpPosix
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jdeStrncmpV)
#   define jdeStrncmpV          jdeStrncmp
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jdeStrncpyV)
#   define jdeStrncpyV          jdeStrncpy
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jdeStrnicmpPosixV)
#   define jdeStrnicmpPosixV    jdeStrnicmpPosix
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jdeStrnicmpV)
#   define jdeStrnicmpV         jdeStrnicmp
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jdeVsnprintfV)
#   define jdeVsnprintfV        jdeVsnprintf
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jdeZStrlenSigSafeV)
#   define jdeZStrlenSigSafeV   jdeZStrlenSigSafe
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jdeZStrlenV)
#   define jdeZStrlenV          jdeZStrlen
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jdeZStrnicmpV)
#   define jdeZStrnicmpV        jdeZStrnicmp
#endif
```

time

The time function will change between 32-bit and 64-bit. The jdetime will standardize these differences.

```
#if !defined(JDENV_64BIT) && !defined(jdetime)
#   define jdetime                time
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jdeAllocV)
#   define jdeAllocV              jdeAlloc
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jdeReAllocExV)
#   define jdeReAllocExV         jdeReAllocEx
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jdemallocV)
#   define jdemallocV            jdemalloc
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jdecallocV)
#   define jdecallocV           jdecalloc
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jdereallocV)
#   define jdereallocV          jderealloc
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jdeAllocReturnIfErrorV)
#   define jdeAllocReturnIfErrorVjdeAllocReturnIfError
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jdeReAllocReturnIfErrorV)
#   define  jdeReAllocReturnIfErrorV  jdeReAllocReturnIfError
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jdeCast16)
```

```
# define jdeCast16(x)          (x)
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jdeCast16U)
# define jdeCast16U(x)        (x)
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jdeCast32)
# define jdeCast32(x)         (x)
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jdeCast32U)
# define jdeCast32U(x)        (x)
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jdeTrunc32)
# define jdeTrunc32(x)         (x)
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(jdeTrunc32U)
# define jdeTrunc32U(x)        (x)
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(TimeToMathNumeric)
# define TimeToMathNumeric      IntToMathNumeric
#endif
```

Named size_t Print and Scan Formats

```
#if !defined(JDENV_64BIT) && !defined(JDEPRISZT)
/* printf of a size_t datatype to a JCHAR string for both 32 & 64-bit values */
# define JDEPRISZT    _J("u")
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(JDESCNSZT)
```

```

/* scanf of a size_t datatype to a JCHAR string for both 32 & 64-bit values */
# define JDESCNSZT _J("u")
#endif

```

```

#if !defined(JDENV_64BIT) && !defined(JDEPRISSZT)
/* printf of a ssize_t datatype to a JCHAR string for both 32 & 64-bit values */
# define JDEPRISSZT _J("d")
#endif

```

```

#if !defined(JDENV_64BIT) && !defined(JDESCNSSZT)
/* scanf of a ssize_t datatype to a JCHAR string for both 32 & 64-bit values */
# define JDESCNSSZT _J("d")
#endif

```

```

#if !defined(JDENV_64BIT) && !defined(JDEZPRISZT)
/* printf of a size_t datatype to a ZCHAR string for both 32 & 64-bit values */
# define JDEZPRISZT _Z("u")
#endif

```

```

#if !defined(JDENV_64BIT) && !defined(JDEZSCNSZT)
/* scanf of a size_t datatype to a ZCHAR string for both 32 & 64-bit values */
# define JDEZSCNSZT _Z("u")
#endif

```

```

#if !defined(JDENV_64BIT) && !defined(JDEZPRISSZT)
/* printf of a ssize_t datatype to a ZCHAR string for both 32 & 64-bit values */
# define JDEZPRISSZT _Z("d")
#endif

```

```

#if !defined(JDENV_64BIT) && !defined(JDEZSCNSSZT)
/* scanf of a ssize_t datatype to a ZCHAR string for both 32 & 64-bit values */
# define JDEZSCNSSZT _Z("d")
#endif

```

Named time_t Print and Scan Formats

```
#if !defined(JDENV_64BIT) && !defined(JDEPRITIM)
/* printf of a time_t datatype to a JCHAR string for both 32 & 64-bit values */
# define JDEPRITIM _J("d")
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(JDESCNTIM)
/* scanf of a time_t datatype to a JCHAR string for both 32 & 64-bit values */
# define JDESCNTIM _J("d")
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(JDEZPRITIM)
/* printf of a time_t datatype to a ZCHAR string for both 32 & 64-bit values */
# define JDEZPRITIM _Z("d")
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(JDEZSCNTIM)
/* scanf of a time_t datatype to a ZCHAR string for both 32 & 64-bit values */
# define JDEZSCNTIM _Z("d")
#endif
```

Named Pointer Print Formats

Printing pointers are necessary because of iSeries performance issues with using %p.

```
#if !defined(JDENV_64BIT) && !defined(JDEPRIPTR)
/* printf of a pointer value to a JCHAR string for all platforms*/
# if defined(JDENV_AS400)
# define JDEPRIPTR _J("#p")
# else
# define JDEPRIPTR _J("p")
# endif
#endif
```

```
#if !defined(JDENV_64BIT) && !defined(JDEZPRIPTR)
```

```

/* printf of a pointer value to a ZCHAR string for all platforms*/
    #if defined(JDENV_AS400)
    # define JDEZPRIPTR _Z("#p")
    #else
    # define JDEZPRIPTR _Z("p")
    #endif

#endif

```

APPENDIX B - 64-BIT NATIVE DATA TYPES BY PLATFORM

Data Type	Windows 32-bit Executable Bytes/bits	Windows 64-bit Executable Bytes/bits	UNIX 32-bit Executable Bytes/bits	UNIX 64-bit Executable Bytes/bits	IBM i 32-bit Compatible Bytes/bits	IBM i 64-bit Compatible Bytes/bits
char	1 / 8	1 / 8	1 / 8	1 / 8	1 / 8	1 / 8
short	2 / 16	2 / 16	2 / 16	2 / 16	2 / 16	2 / 16
int	4 / 32	4 / 32	4 / 32	4 / 32	4 / 32	4 / 32
long	4 / 32	4 / 32	4 / 32	8 / 64	4 / 32	4 / 32
long long	8 / 64	8 / 64	8 / 64	8 / 64	8 / 64	8 / 64
size_t	4 / 32	8 / 64	4 / 32	8 / 64	4 / 32	4 / 32 size64_t = 8/64
off_t	4 / 32	8 / 64	4 / 32	8 / 64	4 / 32	off_t = 4 / 32 off64_t = 8/64
time_t	4 / 32	8 / 64	4 / 32	8 / 64	4 / 32	time_t = 4 / 32 time64_t = 8/64
ptrdiff_t	4 / 32	8 / 64	4 / 32	8 / 64	4 / 32	4 / 32
pointer (void *)	4 / 32	8 / 64	4 / 32	8 / 64	16 / 128	16 / 128

Data Type	AIX 32-bit Executable Bytes/bits	AIX 64-bit Executable Bytes/bits	Other Platforms 32-bit Executable Bytes/bits	Other Platforms 64-bit Executable Bytes/bits
wchar_t	2 / 16	4 / 32	2 / 16	2 / 16

APPENDIX C – COMPILER WARNINGS

This appendix is a quick reference for warning messages that might appear in 64-bit builds.

Simply search for a compiler warning message within the table. A single message may show more than once if there are multiple scenarios.

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
pointer Issue	WIN64 HP64	WIN64: warning C4306: 'type cast' : conversion from 'int' to 'CALLMAP **' of greater size HP64: warning #4231-D: 64 bit migration: conversion between types of different sizes has occurred (from "int" to "CALLMAP **")	<pre>jdeCallObject(_J("GetExchRate") , NULL, lpBhvrCom, lpVoid, &dsGetExch angeRateParms, (CALLMAP*)I DERRszAgreemen tCurrency_13, (int)0, (JCHAR*)NU LL, (JCHAR*)NU LL, (int)0);</pre>	You have a #define integer that is being cast as (CALLMAP*)IDERRszAgreementCurrency_13 . The integer IDERRszAgreementCurrency_13 can never be a pointer value. The function jdeCallObject() requires a pointer to the location of these values, not the values themselves.	<pre>/* 24908308 - Define CALLMAP to map agreement currency to IDERRszCurrencyCodeTo_9 of function GetExchRate */ CALLMAP cmD0000033[1] = { { IDERRszAgreementCurrency_13, IDERRszCurrencyCodeFrom_8 }, }; ... jdeCallObject(_J("GetExchRate"), NULL, lpBhvrCom, lpVoid, &dsGetExchangeRateParms, (CALLMAP*)&cmD0000033, (int)1, (JCHAR*)NULL, (JCHAR*)NULL, (int)0);</pre>	
pointer Issue	WIN64 AIX64 HP64	WIN64: warning C4306: 'type cast' : conversion from 'JDEDB_RESULT' to 'LPDSTMPL' of greater size AIX64: 1506-745 (I) 64-bit portability: possible incorrect pointer through conversion of int type into pointer. HP64: warning #4231-D: 64 bit migration: conversion between types of different sizes has occurred (from "JDEDB_RESULT" to "LPDSTMPL")	<pre>(LPDSTMPL) JDBRS_GetDSTM PLSpecs ((HJDESPEC) NULL, (JCHAR *) szDsTmpl, &lpDSwork- ->lpBlob- ->lpTSDSMPL);</pre>	The result of a function was not assigned to a variable. However, the function still had a preceding cast to a type that was not returned by the function.	<pre>Simply remove the cast. JDBRS_GetDSTMPLSpecs ((HJDESPEC) NULL, (JCHAR *) szDsTmpl, &lpDSwork->lpBlob- ->lpTSDSMPL);</pre>	

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
pointer Issue	WIN64 AIX64 HP64 LNX64	<p>WIN64: warning C4306: 'type cast' : conversion from 'ID' to 'const JCHAR *' of greater size</p> <p>AIX64: 1506-745 (I) 64-bit portability: possible incorrect pointer through conversion of unsigned int type into pointer.</p> <p>HP64: warning #4231-D: 64 bit migration: conversion between types of different sizes has occurred (from "ID" to "const JCHAR *")</p> <p>LNX64: warning: cast to pointer from integer of different size</p>	<pre>ID szEventID ; // Structure member ... jdeStrncpyTerminate(szEventID, (const JCHAR *)dsD0100077.szEventID, DIM(szEventID));</pre>	<p>An ID value is being cast to a const JCHAR *, however, the target variable szEventID was never used subsequently. jdeStrncpyTerminate(szEventID, (const JCHAR *)dsD0100077.szEventID, DIM(szEventID));</p>	<p>Removed to bad line of code...this is not always the Recommendation.</p>	

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
pointer Issue	WIN64 AIX64 HP64 LNX64	<p>WIN64: warning C4305: 'type cast' : truncation from 'LPVOID' to 'int' warning C4306: 'type cast' : conversion from 'int' to 'void *' of greater size</p> <p>AIX64: 1506-744 (I) 64-bit portability: possible truncation of pointer through conversion of pointer type into int type. 1506-745 (I) 64-bit portability: possible incorrect pointer through conversion of int type into pointer.</p> <p>HP64: warning #2767-D: conversion from pointer to smaller integer warning #4231-D: 64 bit migration: conversion between types of different sizes has occurred (from int to void *)</p> <p>LNX64: warning: cast from pointer to integer of different size warning: cast to pointer from integer of different size</p>	<pre>memcpy((void *)(int)(lpUpdateStr ucture) + offset), (const void *)lpTempColumnV alue, lpCurrentCol umn->nLength);</pre>	<p>The objective here is to advance the pointer by a certain number of BYTES prior to performing the memcpy.</p> <p>To add a byte count to the pointer, the pointer lpUpdateStructure must first be cast to a BYTE*. Then the offset (bytes) can be added to obtain the correct pointer location, and finally, the pointer is then cast back to a void* prior to the call to memcpy.</p>	<pre>memcpy((void *)((BYTE*)(lpUpdateStructure)+offset), (const void *)lpTempColumnValue, lpCurrentColumn->nLength);</pre>	
pointer Issue	Win64	<p>warning C4306: 'type cast' : conversion from 'int' to 'JCHAR *' of greater size</p>	<pre>idDBReturn = JDB_OpenTable (hUser, NID_F0101, ID_F0101_A DDRESS, szCol, (ushort) nNumCols, (JCHAR *) JDEDB_COMMIT_ AUTO, &hRequestF0101);</pre>	<p>In the call to JDB_OpenTable, the sixth parameter is supposed to be a pointer. However, here, it is being passed an enum integer.</p>	<pre>idDBReturn = JDB_OpenTable(hUser, NID_F0101, ID_F0101_ADDRESS, szCol, (ushort) nNumCols, (JCHAR *) NULL, &hRequestF0101);</pre>	

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
pointer Issue	WIN64 AIX64 HP64 LNX64	<p>WIN64: warning C4305: 'type cast' : truncation from 'HREQUEST *' to 'unsigned int'</p> <p>warning C4306: 'type cast' : conversion from 'ID' to 'HREQUEST' of greater size</p> <p>AIX64: 1506-744 (I) 64-bit portability: possible truncation of pointer through conversion of pointer type into unsigned int type.</p> <p>1506-745 (I) 64-bit portability: possible incorrect pointer through conversion of unsigned int type into pointer.</p> <p>HP64: warning #2767-D: conversion from pointer to smaller integer</p> <p>warning #4231-D: 64 bit migration: conversion between types of different sizes has occurred (from ID to HREQUEST)</p> <p>LNX64: warning: cast from pointer to integer of different size</p> <p>warning: cast to pointer from integer of different size</p>	<pre>HREQUEST dsF 01133THdl=(HRE QUEST)NULL; HREQUEST *lpF 01133THdl; ... lpDS- >idFileHandle = (unsigned long)(lpF01133TH dl); ... lpF01133THdl = (HREQUEST)lpDS ->idFileHandle;</pre>	<p>HREQUEST *lpF01133THdl is a pointer variable that is being assigned to an ID integral variable. Later, the ID is being used to repopulate the pointer variable. Do not assign a pointer to anything other than another pointer variable. The correct practice is to use the jdeStoreDataPtr / jdeRetrieveDataPtr / jdeRemoveDataPtr functions.</p>	<pre>... lpDS->idFileHandle = jdeStoreDataPtr(hUser, hRequestF01133); ... hRequestF01133 = (HREQUEST) jdeRetrieveDataPtr(hUser, lpDS- >idFileHandle); ... hRequestF01133 = (HREQUEST)jdeRemoveDataPtr(hUser, lpDS->idFileHandle);</pre>	

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
pointer Issue	AIX64 HP64 LNX64 WIN64	<p>AIX64: 1506-745 (I) 64-bit portability: possible incorrect pointer through conversion of unsigned int type into pointer.</p> <p>HP64: warning #4231-D: 64 bit migration: conversion between types of different sizes has occurred (from ID to JCHAR *)</p> <p>LNX64: warning: cast to pointer from integer of different size</p> <p>WIN64: warning C4306: 'type cast' : conversion from 'ID' to 'JCHAR *' of greater size</p>	<pre>typedef struct tagDSD31B0790A { ... ID idErrorMessageID; ... } DSD31B0790A, *LPDSD31B0790A ; ... jdeStrcpy((JCHAR *)(lpDS- >idErrorMessageID), (const JCHAR *)(_J("078N")));</pre>	<p>The objective with this code block is to return an error code Data Dictionary item name back to the user interface layer so that an error can be displayed. In the original code sample, the pointer value is assigned to an ID. Do not assign a pointer to anything other than another pointer variable.</p>	<p>The Recommendation is to add another data item to the datastructure for DD Item to be returned.</p> <pre>typedef struct tagDSD31B0790A { ... ID idErrorMessageID; ... JCHAR szErrorMessageID[11]; } DSD31B0790A, *LPDSD31B0790A; ... jdeStrcpy((JCHAR *)(lpDS- >szErrorMessageID), (const JCHAR *)(_J("078N")));</pre>	
pointer Issue	WIN64 AIX64 HP64 LNX64	<p>WIN64: warning C4305: 'type cast' : truncation from 'void *' to 'ID'</p> <p>AIX64: 1506-744 (I) 64-bit portability: possible truncation of pointer through conversion of pointer type into unsigned int type.</p> <p>HP64 : warning #2767-D: conversion from pointer to smaller integer</p> <p>LNX64: warning: cast from pointer to integer of different size</p>	<pre>lpDS->idhCursor = (ID)jdeRemoveDat aPtr (lpdsInternal- >hUser, (unsigned long)lpDS- >idhCursor);</pre>	<p>jdeRemoveDataPtr is being used incorrectly in this example. jdeRemoveDataPtr returns the pointer values that were just removed. In this example, the lpDS->idhCursor is not subsequently used, so it needs to be reset to 0. Removing the assignment may not be the answer in every Recommendation.</p>	<pre>jdeRemoveDataPtr(lpdsInternal->hUser, lpDS->idhCursor); lpDS->idhCursor = (ID)0;</pre>	

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
pointer Issue	WIN64 AIX64 HP64 LNX64	WIN64: warning C4305: 'type cast' : truncation from 'HREQUEST' to 'unsigned int' AIX64: 1506-744 (I) 64-bit portability: possible truncation of pointer through conversion of pointer type into unsigned int type. HP64 : warning #2767-D: conversion from pointer to smaller integer LNX64: warning: cast from pointer to integer of different size	dsGetColumn.idhRequest = (ID)jdeStoreDataPtr(hUser, (void *)hRequestF40R11); ... jdeRemoveDataPtr(hUser, (unsigned long)hRequestF40R11);	Incorrect implementation of the jdeRemoveDataPtr() function. jdeRemoveDataPtr() requires the ID that was defined by the jdeStoreDataPtr() call. In this case, the original pointer is being used as if it were the ID.	dsGetColumn.idhRequest = (ID)jdeStoreDataPtr(hUser, (void *)hRequestF40R11); ... jdeRemoveDataPtr(hUser, dsGetColumn.idhRequest);	
non-64bit	IBM64	CZM0304 No function prototype given for "ApiUnit_Density_API".	Prototype definition: ApiUnit * ApiUnit_Density_API(); Function definition: ApiUnit * ApiUnit_Density_API({...})	The IBM i C compiler follows the C99 C standard in which every function must have a defined list of parameters. If no parameters are required, then the parameter list is void. In this specific warning, the compiler does not like that the function is defined without a specific variable definition. The warning is issued where the function is used, so the correction must be made to the function where it is defined. b41b0353.c	Prototype definition: ApiUnit * ApiUnit_Density_API(void); Function definition: ApiUnit * ApiUnit_Density_API(void){...} And then make sure that the header that defines the prototype is included in the source... #include <b41b0353.h> /* Bug 25058099 - Add missing include file */	

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
non-64bit	SOL64 HP64	SOL64: warning: implicit function declaration: _TEXT (E_NO_IMPLICIT_DECL_ALLOWED) HP64: warning #2223-D: function "_TEXT" declared implicitly	jdeStrcpy(lpDS->szErrorMessage) D, (const JCHAR *) _TEXT(" "));	The warning states that the function or macro _TEXT is not found in the headers that have been included in the source. Some compilers will create an "implied" interface. However, the build warning levels disallow this. You need to understand if the function is available on the platform. In this case, _TEXT is not valid on any platform except Windows. So either you don't use the _TEXT function or you make it "Windows Only". If the program is supposed to use _TEXT(), then you should not build this on other platforms and you need to change the BSFN OMW record to only build in DEVCLIENT ONLY.	In this example, _TEXT is a Windows-Only function and is not valid on other platforms. The function, however, is intended for all platforms. Therefore, use _J("") instead.	
pointer issue	WIN64 HP64	WIN64: warning C4306: 'type cast' : conversion from 'int' to 'JCHAR **' of greater size HP64 : warning #4231-D: 64 bit migration: conversion between types of different sizes has occurred (from int to JCHAR *)	if (idJDBResult == JDEDB_FAILED) { jdeErrorSet(lpBhvrCom, lpVoid, (ID)0, _J("078S"), (LPVOID)NULL); return ((JCHAR *)ER_ERROR); }	ER_ERROR is an integer and not a character pointer. If the function fails, return NULL.	if (idJDBResult == JDEDB_FAILED) { jdeErrorSet(lpBhvrCom, lpVoid, (ID)0, _J("078S"), (LPVOID)NULL); /* Bug 24949873 - ER_ERROR is an integer and not a character pointer. * If the function fails, return NULL. return ((JCHAR *)ER_ERROR); */ return (JCHAR *)NULL; }	

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
pointer Issue	WIN64 AIX64 HP64 LNX64	<p>WIN64: warning C4305: 'type cast' : truncation from 'void *' to 'ID' warning C4306: 'type cast' : conversion from 'ID' to 'void *' of greater size</p> <p>AIX64: 1506-744 (I) 64-bit portability: possible truncation of pointer through conversion of pointer type into unsigned int type. 1506-745 (I) 64-bit portability: possible incorrect pointer through conversion of unsigned int type into pointer.</p> <p>HP64 : warning #2767-D: conversion from pointer to smaller integer warning #4231-D: 64 bit migration: conversion between types of different sizes has occurred (from ""ID"" to ""void *"")</p> <p>LNX64: warning: cast from pointer to integer of different size warning: cast to pointer from integer of different size</p>	<pre>dsDSD90CB114.id Cursor = (ID)jdeRemoveDataPtr(hUser,lpdsDSD90CB127A->idCursorState); ... lpdsDSD90CB127A->idCursorState = (ID)jdeStoreDataPtr(hUser,(void*)dsDSD90CB114.idCursor);</pre>	<p>In the first example, the results of jdeRemoveDataPtr returns the pointer of the ID that is passed in the parameter list. In the second example, jdeStoreDataPtr requires a pointer to be passed as parm2, but the value is being passed in an ID integer.</p> <p>This function had to be restructured to use the correct values and preserve the intent of the function.</p>	<pre>dsDSD90CB114.idCursor = lpdsDSD90CB127A->idCursorState; ... lpdsDSD90CB127A->idCursorState = dsDSD90CB114.idCursor;</pre>	

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
non-64bit	Win64	warning C4505: 'I0101501_GetLocationServiceEnabled': unreferenced local function has been removed	<pre> Prototype: JDEB FRTN (ID) JDEBFWINAPI UpdateAddressG eoCode (LP BHVRCOM lpBhvrCom, LPVOID lpVoid, LPDSD0101501A lpDS); Function: JDEB FRTN (ID) JDEBFWINAPI UpdateAddressG eoCode (LP BHVRCOM lpBhvrCom, LPVOID lpVoid, LPDSD0101501A lpDS) {...} </pre>	A local function is defined but not used, so the compiler drops it. The function and/or its prototype needs to be removed or commented out.	Delete or comment out the local function and its prototype.	
pointer Issue	WIN64 AIX64 HP64 LNX64	<p>WIN64: warning C4305: 'type cast' : truncation from 'HCACHE' to 'unsigned int'</p> <p>AIX64: 1506-744 (I) 64-bit portability: possible truncation of pointer through conversion of pointer type into unsigned int type.</p> <p>HP64: warning #2767-D: conversion from pointer to smaller integer</p> <p>LNX64: warning: cast from pointer to integer of different size</p>	<pre> dsD0701260A.idI0 6UI003CacheHand le = (unsigned long) lpdsControl- >hI06UI003Cache; </pre>	<p>Rather than assigning a pointer to an ID, use jdeStoreDataPtr and pass the returned ID instead.</p> <p>Then finally, if you use jdeStoreDataPtr, then jdeRemoveDataPtr must be used to release the pointer later.</p>	<pre> dsD0701260A.idI06UI003CacheHandle = (ID) jdeStoreDataPtr(lpdsControl->hUser, lpdsControl->hI06UI003Cache); ... jdeRemoveDataPtr(lpdsControl->hUser, dsD0701260A.idI06UI003CacheHandle); </pre>	

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
pointer Issue	WIN64 AIX64 HP64 LNX64	<p>WIN64: warning C4306: 'type cast' : conversion from 'int' to 'unsigned int *' of greater size warning C4306: 'type cast' : conversion from 'tLocGeoCode' to 'unsigned int *' of greater size</p> <p>AIX64: 1506-745 (I) 64-bit portability: possible incorrect pointer through conversion of int type into pointer. 1506-745 (I) 64-bit portability: possible incorrect pointer through conversion of unsigned int type into pointer.</p> <p>HP64 : warning #4231-D: 64 bit migration: conversion between types of different sizes has occurred (from int to unsigned int *) warning #4231-D: 64 bit migration: conversion between types of different sizes has occurred (from tLocGeoCode to unsigned int *)</p> <p>LNX64: warning: cast to pointer from integer of different size warning: cast to pointer from integer of different size</p>	<pre>if ((unsigned long *)lpdsTempPtrTaxArea->IWorkGeoCode == (unsigned long *)IWorkGeoCode)</pre>	<p>Unnecessary casting and neither side of the relational expression is a pointer. Therefore, casting to a pointer is incorrect.</p>	<pre>if (lpdsTempPtrTaxArea->IWorkGeoCode == IWorkGeoCode)</pre>	

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
pointer Issue	WIN64 AIX64 HP64 LNX64	<p>WIN64: warning C4306: 'type cast' : conversion from 'int' to 'int *' of greater size</p> <p>AIX64: 1506-745 (I) 64-bit portability: possible incorrect pointer through conversion of int type into pointer.</p> <p>HP64 : warning #4231-D: 64 bit migration: conversion between types of different sizes has occurred (from int to int *)</p> <p>LNX64: warning: cast to pointer from integer of different size</p>	<pre>bResult = dsGeoCodeInit.GeoCodeA oCodeAPI.LocSet Attrib (IConnect, eLocAttribUpdtZip 9, (long *) iZip4);</pre>	<p>The dsGeoCodeInit.GeoCodeAPI.LocSetAttrib() function requires a pointer to a long integer. In this case, the address of the iZip4 variable should be passed and not the value.</p>	<pre>bResult = dsGeoCodeInit.GeoCodeAPI.LocSetAttrib (IConnect, eLocAttribUpdtZip9, &iZip4);</pre>	

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
pointer Issue	WIN64 AIX64 HP64 LNX64	<p>WIN64 warning C4305: 'type cast' : truncation from 'LPLINKLIST' to 'unsigned int' warning C4305: 'type cast' : truncation from 'void *' to 'ID'</p> <p>AIX64 1506-744 (I) 64-bit portability: possible truncation of pointer through conversion of pointer type into unsigned int type. 1506-744 (I) 64-bit portability: possible truncation of pointer through conversion of pointer type into unsigned int type.</p> <p>HP64 warning #2767-D: conversion from pointer to smaller integer warning #2767-D: conversion from pointer to smaller integer</p> <p>LNX64 warning: cast from pointer to integer of different size warning: cast from pointer to integer of different size</p>	<pre>lpDS->idPointerToLinkedList = (ID)jdeRemoveDataPtr(hUser, (unsigned long)lplinkListIndex);</pre>	<p>jdeRemoveDataPtr returns a pointer, not an ID. It also takes an ID, not a pointer. Change this line to not save the return value because you are cleaning up and just freeing up the pointer storage location - which means that you need to pass the storage location and not a pointer as the second parameter (which is null at this point because of the line above).</p>	<pre>jdeRemoveDataPtr(hUser, lpDS->idPointerToLinkedList);</pre>	

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
pointer Issue	WIN64 AIX64 HP64 LNX64	<p>WIN64: warning C4306: 'type cast' : conversion from 'int' to 'JCHAR *' of greater size</p> <p>AIX64: 1506-745 (I) 64-bit portability: possible incorrect pointer through conversion of int type into pointer.</p> <p>HP64 : warning #4231-D: 64 bit migration: conversion between types of different sizes has occurred (from int to JCHAR *)</p> <p>LNX64: warning: cast to pointer from integer of different size</p>	<pre>if ((JCHAR*)(lpdsPrOptVersion->szUnitOfMeasureSelection[0] == _J(' ')) (lpdsPrOptVersion->szUnitOfMeasureSelection[0] == _J("\0")))</pre>	<p>lpdsPrOptVersion->szUnitOfMeasureSelection[0] is a single Unicode character of type int, not a pointer.</p>	<pre>if (lpdsPrOptVersion->szUnitOfMeasureSelection[0] == _J(' ') lpdsPrOptVersion->szUnitOfMeasureSelection[0] == _J("\0"))</pre>	
pointer Issue	WIN64 AIX64 HP64 LNX64	<p>WIN64: warning C4305: 'type cast' : truncation from 'NID' to 'ID'</p> <p>AIX64: 1506-744 (I) 64-bit portability: possible truncation of pointer through conversion of pointer type into unsigned int type.</p> <p>HP64 : warning #2767-D: conversion from pointer to smaller integer</p> <p>LNX64: warning: cast from pointer to integer of different size</p>	<pre>NID szFirs tKeyInIndex = _J(""); ... jdeSetGBRError(lpBhvrCom, lpVoid, (ID)szFirstKeyInIndex, _J("0002"));</pre>	<p>jdeSetGBRError wants an ID</p>	<pre>ID idFirstKeyInIndex = (ID)0; ... jdeSetGBRError(lpBhvrCom, lpVoid, idFirstKeyInIndex, _J("0002"));</pre>	

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
pointer Issue	WIN64 AIX64 HP64 LNX64	<p>WIN64: warning C4305: 'type cast' : truncation from 'void *' to 'int' warning C4306: 'type cast' : conversion from 'int' to 'LPVOID' of greater size</p> <p>AIX64: 1506-744 (I) 64-bit portability: possible truncation of pointer through conversion of pointer type into int type. 1506-745 (I) 64-bit portability: possible incorrect pointer through conversion of int type into pointer.</p> <p>HP64 : warning #2767-D: conversion from pointer to smaller integer warning #4231-D: 64 bit migration: conversion between types of different sizes has occurred (from int to LPVOID)</p> <p>LNX64: warning: cast from pointer to integer of different size warning: cast to pointer from integer of different size</p>	<pre>IAddr = (long) lpdsIndexData + lIndexOffSet ; lpmnData1 = (LPVOID) IAddr ;</pre>	<pre>/* Bug 24836807 - Cast the lpdsTableData to a byte * and assigned directly to the lpData variable */</pre>	<pre>lpmnData1 = (BYTE *) lpdsIndexData + lIndexOffSet;</pre>	
pointer Issue	WIN64 AIX64 HP64 LNX64	<p>WIN64: warning C4306: 'type cast' : conversion from 'int' to 'int *' of greater size</p> <p>AIX64: 1506-745 (I) 64-bit portability: possible incorrect pointer through conversion of int type into pointer.</p> <p>HP64 : warning #4231-D: 64 bit migration: conversion between types of different sizes has occurred (from int to int *)</p> <p>LNX64: warning: cast to pointer from integer of different size</p>	<pre>MathNumericToInt (&lpDS- >mnMaxCatalogTr eeLevel, (int *)nMaxCatalogLev el);</pre>	<pre>MathNumericToInt requires a pointer to the Integer value and not the value itself.</pre>	<pre>MathNumericToInt(&lpDS- >mnMaxCatalogTreeLevel, &nMaxCatalogLevel);</pre>	

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
pointer Issue	WIN64 AIX64 HP64 LNX64	<p>WIN64: warning C4305: 'type cast' : truncation from 'HREQUEST' to 'ID'</p> <p>AIX64: 1506-744 (I) 64-bit portability: possible truncation of pointer through conversion of pointer type into unsigned int type.</p> <p>HP64 : warning #2767-D: conversion from pointer to smaller integer</p> <p>LNX64: warning: cast from pointer to integer of different size</p>	<pre>dsWriteCardex.idHRequestF4111 = (ID) hRequestF4111;</pre>	<pre>/* Bug 25036100 - Store the hRequest before passing it. None of the functions * that this is being passed to currently use the value so this will not * break anything. */</pre>	<pre>dsWriteCardex.idHRequestF4111 = jdeStoreDataPtr(hUser, hRequestF4111);</pre>	
pointer Issue	WIN64 AIX64 HP64 LNX64	<p>WIN64: warning C4305: 'type cast' : truncation from 'void *' to 'XAPI_CALL_ID'</p> <p>AIX64: 1506-744 (I) 64-bit portability: possible truncation of pointer through conversion of pointer type into int type.</p> <p>HP64 : warning #2767-D: conversion from pointer to smaller integer</p> <p>LNX64: warning: cast from pointer to integer of different size</p>	<pre>ulXAPICallID = (XAPI_CALL_ID) jdeRetrieveDataPtr(hUser, (unsigned long)lpDS->idXapiCall);</pre>	<p>jdeRetrieveDataPtr returns the pointer that was saved at ID lpDS->idXapiCall. The pointer value cannot be assigned to an ID variable. An ID to an ID is a simple assignment.</p>	<pre>ulXAPICallID = lpDS->idXapiCall;</pre>	

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
pointer Issue	WIN64 AIX64 HP64 LNX64	<p>WIN64: b7601150.c(753) : warning C4306: 'type cast' : conversion from 'ID' to 'void *' of greater size</p> <p>AIX64: 1506-745 (I) 64-bit portability: possible incorrect pointer through conversion of unsigned int type into pointer.</p> <p>HP64 : warning #4231-D: 64 bit migration: conversion between types of different sizes has occurred (from ""ID"" to ""void *"")</p> <p>LNX64: warning: cast to pointer from integer of different size</p>	<pre>if ((void *) dsD4200260.idPtr ToF4211DS != NULL)</pre>	<p>The data type of the variable dsD4200260.idPtrToF4211DS is an integral ID (range of values 0-4GB). The comparison needs to be made against an integral value as well. NULL is reserved for Pointers. It is defined as (void*)0 which is a pointer.</p>	<pre>if (dsD4200260.idPtrToF4211DS != 0)</pre>	
pointer Issue	Win64	<p>warning C4305: 'type cast' : truncation from 'void *' to 'JDEDB_LEVEL'</p>	<pre>idJDEDBReturn = JDEDBRequestID Share(hUser, NID_F0004, ID_F0 004_SYSTEM__T YPE, szCol F0004, nNumColsF0004, JDEDB_COMMIT_ AUTO, JDED B_SHARE_PUBLI C, NULL, &hRe questF0004);</pre>	<p>NULL is an invalid parameter for this function</p>	<pre>idJDEDBReturn = JDEDBRequestIDShare(hUser, NID_F0004, ID_F0004_SYSTEM_TYPE, szColF0004, nNumColsF0004, JDEDB_COMMIT_AUTO, JDEDB_SHARE_PUBLIC, JDEDB_LEVEL_CURRENT, &hRequestF0004);</pre>	

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
time_t issue	WIN64 AIX64 HP64 IBM i SOL64	<p>WIN64: warning C4133: 'function' : incompatible types - from 'const int *' to 'const time_t *'</p> <p>AIX64: 1506-280 (W) Function argument assignment between types "const long*" and "const int*" is not allowed.</p> <p>HP64 : warning #4228-D: 64 bit migration: conversion from "const int *" to a more strictly aligned type "const time_t *" may cause misaligned access</p> <p>IBM i: CZM0280 Function argument assignment between types "const long long*" and "const int*" is not allowed.</p> <p>SOL64: "warning: argument #1 is incompatible with prototype:</p> <p>LNx64: warning: passing argument 1 of 'jdeGmtime' from incompatible pointer type</p>	<pre>jdeGmtime((const int *)&IUTCTime, &zTime);</pre>	<p>jdeGmtime requires a time_t value</p>	<p>Change &IUTCTime to be time_t datatype and remove the (const long *) cast</p>	<p>Change &IUTCTime to be jdetime_t datatype and remove the (const int *) cast</p>

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
time_t issue	WIN64 AIX64 HP64 LNX64 SOL64	WIN64: warning C4133: 'function' : incompatible types - from 'int **' to 'const time_t *' AIX64: 1506-280 (W) Function argument assignment between types const long* and int* is not allowed. HP64 : warning #4228-D: 64 bit migration: conversion from int * to a more strictly aligned type const time_t * may cause misaligned access LNX64: warning: passing argument 1 of 'jdeLocaltime' from incompatible pointer type SOL64: warning: argument #1 is incompatible with prototype:	int ITime; jdeLocaltime(&ITime, &dsLocalTime);	jdeLocaltime requires a jdetime_t (time_t) datatype for variable used as parm #1 of the function call.	time_t ITime; jdeLocaltime(&ITime, &dsLocalTime);	jdetime_t ITime; jdeLocaltime(&ITime, &dsLocalTime);
time_t issue	WIN64	warning C4242: '=' : conversion from 'time_t' to 'int', possible loss of data	int ITime; /*A temp field for use with get time*/ ITime = jdeTime(NULL);	In 32-bit, time_t will remain 32-bit. In 64-bit, time_t will become a 64-bit value to accommodate the time() values after the year 2038. The time() function is renamed to jdetime() and time_t is renamed to jdetime_t to be consistent across all supported platforms.	time_t tTime; /*A temp field for use with get time*/ tTime = time(NULL);	jdetime_t tTime; /*A temp field for use with get time*/ tTime = jdetime(NULL);
jdeStrlen()	32-bit unsigned value	jdeStrlenV()	64-bit unsigned value	jdeStrlen()	32-bit unsigned value	jdeStrlenV()
non-64bit	WIN64	warning C4005: 'WS_VISIBLE' : macro redefinition	Duplicate definition to one defined by Windows winuser.h	Remove the #define statement that's duplicated.	Remove the #define statement that's duplicated	
non-64bit	WIN64	warning C4142: benign redefinition of type	typedef long Trip;	The data type is defined more than once.	Remove the duplicate definition	

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
non-64bit	WIN64	warning C4505: 'I0101501_GetLocationServiceEnabled': unreferenced local function has been removed	static ID I0101501_GetLocationServiceEnabled(LPBHVRCOM lpBhvrCom, LPVOID lpVoid, JCHAR cLocationServiceEnabled); static ID I0101501_GetLocationServiceEnabled(LPBHVRCOM lpBhvrCom, LPVOID lpVoid, JCHAR cLocationServiceEnabled) {...}	The local function is no longer used and needs to be removed from the source file.	Either delete the code or comment it out. Note that this must be done for both the prototype in the object.h file and the function definition in the object.c file.	
non-64bit	SOL64	warning: enum type mismatch: op "!=" (E_ENUM_TYPE_MISMATCH_OP)	if (eDMStatus != DATAMAPCDIST_SUCCESS)	DATAMAPCDIST_SUCCESS is not a member of the enum type that defines the eDMStatus variable.	Recommendation will vary according to the enum and variable data type. In this case, you change the enum datatype to use the correct value. if (eDMStatus != DATAMAP_SUCCESS)	
non-64bit	SOL64	warning: enum type mismatch: op "==" (E_ENUM_TYPE_MISMATCH_OP)	if (eJDBReturn == JDEDB_FAILED)	JDEDB_FAILED is not a member of the enum type that defines the eJDBReturn variable.	Recommendation will vary according to the enum and variable data type. In this case, you change the variable data type to match the return code of the function. if (jJDBReturn == JDEDB_FAILED)	
non-64bit	SOL64	warning: implicit function declaration: foo (E_NO_IMPLICIT_DECL_ALLOWED)	myresult = foo();	Function foo() has not been defined as a prototype in the current source member or in one of the headers that have been included. #include the appropriate header file.		

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
pointer Issue	AIX64 HP64 LNX64 WIN64	<p>AIX64: 1506-745 (I) 64-bit portability: possible incorrect pointer through conversion of int type into pointer.</p> <p>HP64 : warning #4231-D: 64 bit migration: conversion between types of different sizes has occurred (from "int" to "int *")</p> <p>LNX64: warning: cast to pointer from integer of different size</p> <p>WIN64: warning C4133: 'function' : incompatible types - from 'int **' to 'LONG64 *'</p>	<pre>int ISetting = 1; /*13867131*/ ... bReturnValue = dsGeoCodeInit.GeoCodeAPI.LocSet Attrib (IConnect, eLocLastAttrib, (int *) ISetting);</pre>	<p>B0500543 include B0500543.h which includes b0701230.h which includes B0700058.h where LocSetAttrib function is defined as PORT_PREFIX BOOL POSTFIX LocSetAttrib (tLocConn pConn, tLocAttrib pAttrib, PORT_VTXLONG pSetting);</p> <p>PORT_VTXLONG is the data type (actually a #define) that is used for all of the Vertex Geocode function calls.</p> <p>2 problems here: 1) The incorrect cast causes warnings on all Unix and Windows 64-bit platforms. In the call to LocSetAttrib(), ISetting variable needs to be an integral value (long or int) and not a pointer (long *). 2) Since the function expects a PORT_VTXLONG for parm 3, the local variable ISetting technically should be declared as PORT_VTXLONG ISetting = 1; and not long ISetting = 1; PORT_VTXLONG is defined to be 32-bit or 64-bit depending on the compile mode.</p> <p>The long data type will be converted to an int during conversion to 64-bit while PORT_VTXLONG is properly defined for 32-bit vs 64-bit implementation.</p>	<p>First, research the actual VERTEX API and verify that PORT_PREFIX BOOL POSTFIX LocSetAttrib (tLocConn pConn, tLocAttrib pAttrib, PORT_VTXLONG pSetting); is the correct API. Is it "PORT_VTXLONG pSetting" for parm 3 or "PORT_VTXLONG * pSetting"</p> <p>Assuming the API is correct... PORT_VTXLONG ISetting = 1; ... bReturnValue = dsGeoCodeInit.GeoCodeAPI.LocSetAttrib (IConnect, eLocLastAttrib, ISetting);</p>	

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
pointer Issue	WIN64	warning C4133: 'function' : incompatible types - from 'int *' to 'LONG64 *'	<pre>int iZip4 = 1; ... bResult = dsGeoCodeInit.GeoCodeAPI.LocSetAttrib (IConnect, eLocAttribUpdtZip9, &iZip4);</pre>	<p>B0701230.c includes b0701230.h which includes B0700058.h where LocSetAttrib function is defined as</p> <pre>PORT_PREFIX BOOL POSTFIX LocSetAttrib (tLocConn pConn, tLocAttrib pAttrib, PORT_VTXLONG pSetting);</pre> <p>PORT_VTXLONG is the data type (actually a #define) that is used for all of the Vertex Geocode function calls.</p> <p>2 problems here: 1) Parm 3 does not want a pointer. In the call to LocSetAttrib(), parm 3 needs to be an integral value (long or int) and not a pointer (&iZip4). 2) Since the function expects a PORT_VTXLONG for parm 3, the local variable ISetting technically should be declared as PORT_VTXLONG ISetting = 1; and not long ISetting = 1; PORT_VTXLONG is defined to be 32-bit or 64-bit depending on the compile mode.</p> <p>The long data type will be converted to an int during conversion to 64-bit while PORT_VTXLONG is properly defined for 32-bit vs 64-bit implementation.</p>	<p>First, research the Actual VERTEX API and verify that PORT_PREFIX BOOL POSTFIX LocSetAttrib (tLocConn pConn, tLocAttrib pAttrib, PORT_VTXLONG pSetting); is the correct API. Is it "PORT_VTXLONG pSetting" for parm 3 or "PORT_VTXLONG * pSetting"</p> <p>Assuming the API is correct... PORT_VTXLONG iZip4 = 1; ... bResult = dsGeoCodeInit.GeoCodeAPI.LocSetAttrib (IConnect, eLocAttribUpdtZip9, iZip4);</p>	

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
Platform API	WIN64	warning C4242: '=' : conversion from 'INT_PTR' to 'int', possible loss of data	<pre>HINSTANCE hLib = NULL; FARPROC lpFunc = NULL; BOOL bReturn = TRUE; hLib = LoadLibrary(_J("d wcheck.dll")); if (hLib != NULL) { jdeStrcpy(szFunc c, _J("CheckLicense")); lpFunc = GetProcAddress(h Lib, szFunc); if (lpFunc != NULL) bReturn = lpFunc(); }</pre>	<p>FARPROC datatype is defined in WinDef.h as</p> <pre>typedef INT_PTR (FAR WINAPI *FARPROC)();</pre> <p>So FARPROC is a pointer to a function that returns an INT_PTR value.</p> <p>Therefore, in the statement</p> <pre>bReturn = lpFunc();</pre> <p>bReturn needs to be defined as INT_PTR and not BOOL</p>	<pre>HINSTANCE hLib = NULL; FARPROC lpFunc = NULL; INT_PTR bReturn = TRUE; hLib = LoadLibrary(_J("dwcheck.dll")); if (hLib != NULL) { jdeStrcpy(szFunc, _J("CheckLicense")); lpFunc = GetProcAddress(hLib, szFunc); if (lpFunc != NULL) bReturn = lpFunc(); }</pre>	
Platform API	WIN64	warning C4057: 'function' : 'LPINT' differs in indirection to slightly different base types from 'LONG *'	<pre>LOGFONT zLog Font; ... MathNumericToLo ng(&lpDS- >mnCharacterHeig ht, &zLogFont.IfHeight); MathNumericToLo ng(&lpDS- >mnCharacterWei ght, &zLogFont.IfWeigh t);</pre>	<p>LOGFONT is a Windows-only datastructure where the lfHeight and lfWeight members are defined as Windows as LONG (long). In the conversion from 32 to 64-bit source, the conversion tool will change all MathNumericToLong() functions calls to MathNumericToInt(). This is by design; JDE is changing all longs to ints within all BSFN code to ease the conversion to 64-bit for all platforms.</p> <p>In this case, introducing local variables can help transition the values from int to Windows LONG structure members.</p>	<pre>LOGFONT zLogFont; int lHeight=0; int lWeight=0; ... MathNumericToInt(&lpDS- >mnCharacterHeight, &lHeight); MathNumericToInt(&lpDS- >mnCharacterWeight, &lWeight); zLogFont.IfHeight =lHeight; zLogFont.IfWeight =lWeight;</pre>	

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
Platform API	WIN64	warning C4242: '=' : conversion from 'intptr_t' to 'int', possible loss of data	<pre> long hFile e = 0; ... hFile = _tfindfirst(szFileSpec, &c_file) </pre>	<p>This code builds cleanly in 32-bit; however, both 'long' and 'int' will fail in 64-bit mode.</p> <p>_tfindfirst and its sister functions, _findnext and _findclose, return or use a 'intptr_t' value as a handle. 'intptr_t' is an unsigned integer the size of a pointer. With this data type, the size of hFile will flex according to the compiler mode. Therefore, neither 'long' nor 'int' will work as both these types will be 32-bits on Windows 64-bit mode. The proper Recommendation is to use the 'intptr_t' type as the file handle.</p>	<pre> intptr_t hFile = 0; ... hFile = _tfindfirst(szFileSpec, &c_file) </pre>	

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
pointer issue	WIN64	warning C4242: '=' : conversion from 'intptr_t' to 'ID', possible loss of data	<pre>typedef struct tagDSD9600471B { ID hFile; ... //plus others } DSD9600471B, *LPDSD9600471B ; ... if((lpDS->hFile = _tfindfirst(lpDS- >szTargetDirectory , &c_file)) != -1L) {...}</pre>	<p>_tfindfirst and its sister functions, _findnext and _findclose, return or use a 'intptr_t' value as a handle. 'intptr_t' is an unsigned integer the size of a pointer. The intptr_t handle is being assigned to a data structure member where it can be used in other locations of the system. It is a pointer and not an intptr_t integral. This is another example of a pointer assigned to an ID. The correct practice is to use the jdeStoreDataPtr / jdeRetrieveDataPtr / jdeRemoveDataPtr functions to store pointer values.</p>	<pre>In function FindFirstFileMatchingWildcard(): //no change to lpDS structure. intptr_t iptrHFile = 0; ... lpDS->hFile = 0; if ((iptrHFile = _tfindfirst(lpDS- >szTargetDirectory, &c_file)) != -1L) { lpDS->hFile = (ID)jdeStoreDataPtr(hUser, (void *)iptrHFile); ... } ----- In function FindNextFileMatchingWildcard(): intptr_t iptrHFile = 0; iptrHFile = (intptr_t) jdeRetrieveDataPtr(hUser, lpDS->hFile); if (_tfindnext(iptrHFile, &c_file) == 0) {...} ----- in function FindCloseFileMatchingWildcard(): intptr_t iptrHFile = 0; iptrHFile = (intptr_t) jdeRemoveDataPtr(hUser, lpDS->hFile); if (_findclose(iptrHFile) == 0) {...}</pre>	
non-64bit	WIN64	warning C4127: conditional expression is constant	<pre>while(TRUE){...}</pre>	<p>Sometimes an infinite loop is a necessary construction in code. C prefers that you use a for loop for this purpose</p>	<pre>for(;;) {...}</pre>	

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
non-64bit	WIN64	warning C4057: 'function' : 'ulong *' differs in indirection to slightly different base types from 'ID *'	<pre> jdeSpecGetCount(lpds->hSpec, NULL, 0, &lpds->ds9600188A->idNumberOfSpecRecords); </pre>	<p>jdeSpecGetCount() is defined with the 4th parameter as ulong*. KRNL_RTN(JDESPEC_RESULT) JDEWINAPI jdeSpecGetCount(HJDESPEC hSpec , void* pKeyStruct , int iElementCount , ulong* pRecordCount);</p> <p>idNumberOfSpecRecords is defined as an ID. Both ulong and ID are natively defined as "unsigned int", so in normal circumstances, this may work. However, Windows is particularly concerned about the data type of pointer values that are passed to the function and warns if the variable data type name (ID *) is not the same as the function prototype (ulong*) . Since ulong and ID are otherwise identical, you are safe to cast the 4th parameter.</p>	<pre> jdeSpecGetCount(lpds->hSpec, NULL, 0, (ulong *) &lpds->ds9600188A->idNumberOfSpecRecords); </pre>	

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
non-64bit	WIN64	warning C4090: 'function' : different 'const' qualifiers	<pre> IpDDText = jdeAllocFetchDDTextFrom extFromDDItemNameOvr (lpDictionary, lpdsCache->szDataItem, ttD DictTextType, (const JCHAR *) lpDS->szLanguagePreference, (const JCHAR *) _J(" ")); </pre>	<p>The jdeAllocFetchDDTextFrom DDItemNameOvr() prototype is defined as KRNL_RTN(LPDDTEXT) JDEWINAPI jdeAllocFetchDDTextFrom DDItemNameOvr(LPDICTIONARY lpDictionary, PJSTR lp.szItemName, TEXTTYPE iTextType, JCHAR * pszLanguage, JCHAR * pszSystemCode);</p> <p>The pszLanguage and pszSystemCode params are not "const". Therefore, using the const qualifier has no effect on the passing of these parameters. Remove the const qualifier from the cast.</p>	<pre> IpDDText = jdeAllocFetchDDTextFromDDItemNameOvr (lpDictionary, lpdsCache->szDataItem, ttDDictTextType, (JCHAR *) lpDS->szLanguagePreference, (JCHAR *) _J(" ")); </pre>	

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
Platform API	WIN64	warning C4305: 'type cast' : truncation from 'HINSTANCE' to 'int'	<pre> hInstance = FindExecutable(lp DS- >szFileorPathToFi nd, NULL, lpDS- >szFoundFileorPat h); if((int)hInstance <= 32) { idReturn = ER_ERROR; } </pre>	<p>Some Windows APIs return a HINSTANCE, but it is not a real handle. The specific functions documented by Windows MSDN have unique interface requirements.</p> <p>The warning cannot be removed because the code cannot be changed even in 64-bit. Use #pragma instead to disable the warning around the specific line that has the warning.</p> <p>Problem line of code.</p>	<pre> hInstance = FindExecutable(lpDS- >szFileorPathToFind, NULL, lpDS- >szFoundFileorPath); /* The HINSTANCE return code must be cast to an int even in 64-bit. */ #ifdef JDENV_PC #pragma warning(push) #pragma warning(disable : 4305) #endif /* (int)hInstance gives a "warning C4305: 'type cast' : truncation from 'HINSTANCE' to 'int'" * warning in 64-bit. The HINSTANCE return code must be cast to an int even in 64-bit * according to the Microsoft documentation */ if((int)hInstance <= 32) { idReturn = ER_ERROR; } #ifdef JDENV_PC #pragma warning(pop) #endif </pre>	

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
Integer type mismatch	AIX64	1506-743 (I) 64-bit portability: possible change of result through conversion of int type into unsigned long int type.	<pre>int nSize = 15 + 1; ... jdeGetHostName((JCHAR *)&szServerName, nSize, 1);</pre>	<p>jdeGetHostName function expects a jde_n_charV (an unsigned 64-bit int) data type to be passed to jdeGetHostName().</p> <p>The problem is that the int is a signed, 32-bit value while the function expects an unsigned, 64-bit value. This is a change of both signage and length which throws warnings in AIX.</p> <p>The passed variable must be changed so that it is either unsigned or 64-bit width, or both.</p>	<pre>unsigned int nSize = 15 + 1; ... jdeGetHostName((JCHAR *)&szServerName, nSize, 1);</pre>	
Integer type mismatch	AIX64	<p>1506-742 (I) 64-bit portability: possible loss of digits through conversion of long int type into unsigned int type.</p> <p>1506-743 (I) 64-bit portability: possible change of result through conversion of unsigned int type into long int type.</p>	<pre>jdePPSRand((unsigned int)time((time_t *)NULL));</pre>	<p>jdePPSRand function expects a jdatetime_t (signed 64-bit int in 64-bit) data type to be passed. jdatetime() returns is a signed, 64-bit value in 64-bit mode. Both of these warnings are the result of invalid casting.</p> <p>The cast to (unsigned int) and then passing this to jdePPSRand() causes both problems.</p>	<pre>jdePPSRand(time(NULL));</pre>	<pre>jdePPSRand(jdatetime(NULL));</pre>

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
Integer type mismatch	AIX64	1506-743 (I) 64-bit portability: possible change of result through conversion of unsigned int type into long int type.	<pre>time_t lTime = 0; ... time(&lTime); lTime = lTime & 0xFFFFFFFF;</pre>	<p>A hex value such as 0xFFFFFFFF defaults to an unsigned value. In a bit-wise operation, the signage should match. Cast the hex value 0xFFFFFFFF to a signed data type to produce the correct signage for the target variable.</p> <p>The target variable is time_t. This value is a signed 64-bit integral in 64-bit builds and will not truncate the result.</p> <p>Cast the hex value to a signed 32-bit integral such as (int)0xFFFFFFFF.</p>	<pre>time_t lTime = 0; ... time(&lTime); lTime = lTime & (int)0xFFFFFFFF;</pre>	<pre>jdetime_t lTime = 0; ... jdetime(&lTime); lTime = lTime & (int)0xFFFFFFFF;</pre>
Integer type mismatch	AIX64	1506-743 (I) 64-bit portability: possible change of result through conversion of unsigned int type into long int type.	<pre>jdetime_t lTime = 0; int iTime = 0; /* Convert seconds since 1/1/70 00:00:00 to 8 character hexadecimal string */ jdetime(&lTime); iTime = lTime & 0xFFFFFFFF; /* Make lTime 32-bit regardless of platform */ jdeSprintf(szSessionID, _J("%0.8IX"), iTime);</pre>	<p>This is a similar problem to that of B31B6101 except that the target variable is intentionally 32-bits. The BSFN programs only want the lower 32-bits of the value. Use jdeTrunc32() to conform the value to 32-bits. jdeTrunc32() differs from jdeCast32() in that a warning message is not issued when the value exceeds INT_MAX.</p> <p>jdeTrunc32U() is also available if the result needs to be unsigned. A warning message is not issued when the value exceeds UINT_MAX.</p>	<pre>time_t lTime = 0; int iTime = 0; /* Convert seconds since 1/1/70 00:00:00 to 8 character hexadecimal string */ time(&lTime); iTime = jdeTrunc32(lTime & (int)0xFFFFFFFF); /* Make lTime 32-bit regardless of platform */ jdeSprintf(szSessionID, _J("%0.8IX"), iTime);</pre>	

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
Integer type mismatch	Win64	warning C4267: '=' : conversion from 'size_t' to 'int', possible loss of data	<pre>int IFileLen = 0; ... IBytesRead = fread(lpFileContents, 1, IFileLen, pFile);</pre>	<p>fread function expects a size_t (unsigned 64bit int in 64-bit builds) but is passed to an integer (signed 32bit int).</p> <p>The problem is that the int is a signed, 32-bit value while the function expects an unsigned, 64-bit value. This is a change of both signage AND length which throws warnings in AIX. The passed variable must be changed so that it is either an unsigned int or a 64-bit width int, or both</p>	<pre>size_t IFileLen = 0; ... IBytesRead = fread(lpFileContents, 1, IFileLen, pFile);</pre>	
pointer math	AIX64 HP64 WIN64	1506-742 (I) 64-bit portability: possible loss of digits through conversion of long int type into int type.	<pre>int nOffset = 0; ... nOffset = lpszNew - lpszOld; jdeStrncpy (szString, lpszOld, nOffset);</pre>	<p>This is a pointer math problem. In 32-bit, the difference of two pointers is a 32-bit width value (typically an int data type). In 64-bit, however, it will be a signed 64-bit value ('_int64' or a 'long') and can no longer be assigned to a 32-bit width variable such as an int. Also, the formula cannot be passed as the value of a function call that expects a 32-bit value.</p> <pre>jdeStrncpy (szString, lpszOld, p2-p1);</pre>	<p>There are several ways to handle this problem. This math requires the use of a Forward Compatible Recommendation.</p>	<p>There are several ways to handle this problem. This may require the use of a Forward Compatible Recommendation.</p>

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
Integer type mismatch	AIX64	1506-743 (I) 64-bit portability: possible change of result through conversion of short type into unsigned long int type.	<pre>short nSizeOfDataStructure; ... jdeMemset((void *)GenericLpDSKey, (int) _J('\0'), nSizeOfDataStructure);</pre>	<p>memset along with many other string.h functions accepts a size_t for the number of bytes in the function call. jdeMemset, JDE's JCHAR version of the same function, also expects a size_t-based data type.</p> <p>Size_t is unsigned 32-bit integral in 32-bit builds and unsigned 64-bit integral in 64-bit builds.</p> <p>A short is a signed 16-bit value on all platforms in both 32-bit and 64-bit modes.</p> <p>Most compilers will allow a signed short to be assigned to an unsigned int (32-bit). However, AIX will not allow the change in signage (signed to unsigned) and the change in data type width (16-bits to 64-bits) at the same time.</p> <p>The data structure size variable nSizeOfDataStructure would never be negative; it is recommended that it be changed to an unsigned integral data type such as size_t, unsigned int, or unsigned short.</p>	<pre>size_t nSizeOfDataStructure; ... jdeMemset((void *)GenericLpDSKey, (int) _J('\0'), nSizeOfDataStructure);</pre>	<pre>jdesize_t nSizeOfDataStructure; ... jdeMemset((void *)GenericLpDSKey, (int) _J('\0'), nSizeOfDataStructure); --OR-- jdesizeV_t nSizeOfDataStructure; ... jdeMemsetV((void *)GenericLpDSKey, (int) _J('\0'), nSizeOfDataStructure);</pre>

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
non-64bit	HP64	"b7900010.i", line 66434, procedure B7900010_ImportHTML: warning #20206-D: Possible out of bound access (In expression "fgets((char*)szPTDATA, 60001, fp)", variable "szPTDATA" [b7900010.i:66399] (type: char [30001]) has byte range [0 .. 30000], writing byte range [0 .. 60000].)	<pre> JCHAR sz Line[30001] = {0}; ZCHAR sz PTDATA[30001] = {0}; FILE *fp = 0L; ... while (jdeZFgets((void*) szPTDATA, sizeof(szLine)-1, fp)) { ... } </pre>	<p>The HPUX warning indicates that the memory write operation will overflow the target write location, which means that more characters will be written than the target location is allocated for. This is a serious memory problem.</p> <p>The sizeof(szLine) operation gets the number of bytes of the JCHAR array (60002) and (jdeZFgets((void*) szPTDATA, sizeof(szLine)-1, fp) writes those characters to an array that is 30001 bytes(chars).</p>	<p>There can be several ways to solve this type of problem depending on what the code is designed to do. In this case, the assumption is that jdeZFgets is reading ASCII text, so you need to read the number of bytes based on the target array size.</p> <pre> JCHAR szLine[30001] = {0}; ZCHAR szPTDATA[30001] = {0}; FILE *fp = 0L; ... while (jdeZFgets((void*) szPTDATA, sizeof(szPTDATA)-1, fp)) { ... } </pre>	

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
Integer type mismatch	HP64	warning #4229-D: 64-bit migration: conversion from "off_t" to "int" may truncate value	<pre> int nFileLen = 0; struct jdeZStat info; ... jdeStat(fromFully QualifiedSourceFil eName, &info); nFileLen = info.st_size; if(nFileLen == -1) { .. } ... pBuffer = (BYTE*) jdeAlloc(COMMON _POOL, nFileLen, MEM_ZEROINIT); </pre>	<p>The definition of st_size in the stat structure is: off_t st_size; /* total size, in bytes */</p> <p>off_t will be signed 64-bits on most platforms. Therefore, it cannot be assigned to an int or the value will be truncated.</p> <p>Two options: If the size of the file is never going to be larger than 2GB (the max value of an int variable), then continue using a 32-bit value and use jdeCast32() to downsize the st_size value to 32-bits in a 64-bit build.</p> <p>If a value larger than 2GB is possible (such as the size of a very large text file), then the wider variable size (64-bits) is needed. This means that you have to use flexible "Forward Compatible" variables and functions to support the possible 64-bit Recommendation. In this case, you can use jdesizeV_t to use a "forward compatible" signed 64-bit variable for the file size and then use the V functions that are also "forward compatible" and support 64-bit with inputs.</p>	<p>If info.st_size is < 2GB then... Header: <pre> #if !defined(JDENV_64BIT) && !defined(jdeCast32) # define jdeCast32(x) (x) #endif </pre> Source: <pre> int nFileLen = 0; struct jdeZStat info; ... jdeStat(fromFullyQualifiedSourceFileNam e, &info); nFileLen = jdeCast32(info.st_size); ----- ----- </pre> <p>If info.st_size can be > 2GB then... Header: Source: <pre> jdesizeV_t nFileLen = 0; struct jdeZStat info; ... jdeStat(fromFullyQualifiedSourceFileNam e, &info); nFileLen = info.st_size; if(nFileLen == -1) { .. } ... pBuffer = (BYTE*) jdeAllocV(COMMON_POOL, nFileLen, MEM_ZEROINIT); </pre> </p> </p>	

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
non-64bit	HP64	warning #4274-D: comparison of pointer with integer zero	<pre>if (hUser > (HUSER)NULL) {..}</pre>	<p>The C Standard says that a pointer is either NULL or not NULL. Therefore, if (hUser > NULL) is technically incorrect (but works fine). The correct syntax is either:</p> <p>if (hUser != NULL) -or- if (hUser == NULL)</p>	<pre>if (hUser != (HUSER)NULL) {...}</pre>	
Arithmetic truncation	HP64	warning #4298-D: addition result could be truncated before cast to bigger sized type	<pre>struct tm tp = {0}; time_t tToday; ... tToday = time(NULL); jdelocaltime(&tToday, &tp); tToday = ((long)tp.tm_hour * 10000) + ((long)tp.tm_min * 100) + ((long)tp.tm_sec); LongToMathNumeric((long)tToday, &dsF00165Audit.g dtday);</pre>	<p>The error is occurring with the following line of code, where an hhmss time value is being calculated and saved back into the time_t variable tToday. After the code is converted to 64-bit, the new line of code has 'long's changed to 'int's and becomes :</p> <pre>tToday = ((int)tp.tm_hour * 10000) + ((int)tp.tm_min * 100) + ((int)tp.tm_sec);</pre> <p>There are several ways to resolve this. The simplest recommendation that navigates all issues is to assign the result calculation to a 'long' variable instead of assigning back to the 'time_t' variable. The 'long' variable will then match the intermediate values.</p>	<pre>struct tm tp = {0}; time_t tToday; long lToday; ... tToday = time(NULL); jdelocaltime(&tToday, &tp); lToday = ((long)tp.tm_hour * 10000) + ((long)tp.tm_min * 100) + ((long)tp.tm_sec); LongToMathNumeric(lToday, &dsF00165Audit.gdtday);</pre>	<pre>struct tm tp = {0}; jdetime_t tToday; int iToday; ... tToday = jdetime(NULL); jdelocaltime(&tToday, &tp); iToday = ((int)tp.tm_hour * 10000) + ((int)tp.tm_min * 100) + ((int)tp.tm_sec); IntToMathNumeric((int)iToday, &dsF00165Audit.gdtday);</pre>

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
Function Name Decoration	All	No compiler warning message	<pre> #ifdef JDENV_PC jdeStrcpy(szXMLAPI, _J("_xmlReq_convertFromUTF8ToUnicode@12")); #else jdeStrcpy(szXMLAPI, _J("xmlReq_convertFromUTF8ToUnicode")); #endif </pre>	<p>Dynamic Windows 32-bit function calls require a decorated function name. In this example, the actual function name is "xmlReq_convertFromUTF8ToUnicode". However, the Windows 32-bit function name string is "_xmlReq_convertFromUTF8ToUnicode@12".</p> <p>32-bit Windows requires a leading "_" and a trailing "@###" where ### is the sum of bytes used for the function parameters. 64-bit Windows does not require this decoration and uses just the actual function name. 64-bit Windows now behaves the same as in the Unix and IBM i platforms.</p> <p>The recommendation is to change the Windows-ONLY clause to also exclude:</p> <pre> #ifdef JDENV_PC && !defined(JDENV_64BIT) ...Windows 32-bit function string #else ...All 64-bit and 32-bit non-Windows strings #endif </pre>	<pre> #ifdef JDENV_PC && !defined(JDENV_64BIT) jdeStrcpy(szXMLAPI, _J("_xmlReq_convertFromUTF8ToUnicode@12")); #else jdeStrcpy(szXMLAPI, _J("xmlReq_convertFromUTF8ToUnicode")); #endif </pre>	

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
Function Name Decoration	All	No compiler warning message	<pre>#ifndef JDENV_PC jdeStrcpy(szLib, _J("XercesWrapper.dll")); jdeStrcpy(szFunc, _J("_XRCS_parseXMLStringRemoveEncodingZ@12")); #endif</pre>	Same as previous example with a slightly different Recommendation.	<pre>#ifndef JDENV_PC jdeStrcpy(szLib, _J("XercesWrapper.dll")); #if !defined(JDENV_64BIT) jdeStrcpy(szFunc, _J("_XRCS_parseXMLStringRemoveEncodingZ@12")); #else jdeStrcpy(szFunc, _J("XRCS_parseXMLStringRemoveEncodingZ")); #endif #endif</pre>	
Deprecated Code	ALL	error #2020: identifier "GPoolCommon" is undefined	<pre>/*Initialize the JDE memory pool management utility*/ GPoolCommon = jdeMemoryManagementInit();</pre>	<p>GPoolCommon has been deprecated. In 64-bit builds, the jdeMemoryManagementInit() function is no longer available. It is only maintained in 32-bit builds for backward compatibility.</p> <p>Comment out or delete this line.</p>	<pre>/*Initialize the JDE memory pool management utility*/ /* GPoolCommon = jdeMemoryManagementInit(); */</pre>	

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
64-bit Tooling	All	No compiler warning message	<pre>#define BIN32_ACTIVCONSOLE _J("Bin32\\activconsole.exe")</pre>	<p>Code that contains a reference to one of the following JDE folder names:</p> <pre>system\bin32 system\lib32 system\libv32 <pathcode>\bin32 <pathcode>\BusObj <pathcode>\include <pathcode>\lib32 <pathcode>\obj <pathcode>\source</pre> <p>... needs to use the new 64-bit versions of these names instead in a 64-bit build.</p> <pre>system\bin64 system\lib64 system\libv64 <pathcode>\bin64 <pathcode>\BusObj64 <pathcode>\include64 <pathcode>\lib64 <pathcode>\obj64 <pathcode>\source64</pre>	<p>Need to use the correct folder names according to the BUILD parameters:</p> <pre>#if !defined(JDENV_64BIT) #define BIN_ACTIVCONSOLE _J("bin32\\activconsole.exe") #else #define BIN_ACTIVCONSOLE _J("bin64\\activconsole.exe") #endif ... if (bReturn) { jdePrintf(szActivConsole, _J("%s\\%s"), szSysPath, BIN_ACTIVCONSOLE); }</pre> <p>Notice that BIN32_* was renamed to BIN_* in this example to reflect the change in purpose of the definition.</p> <p>Specific implementation of this example could vary depending on the needs of the BSFN Application.</p>	

Integer type mismatch	AIX64	1506-743 (I) 64-bit portability: possible change of result through conversion of int type into unsigned long int	<pre> int nSize = 0; ... MathNumericToInt (&lpDS- >mnFileChunkSize , &nSize); ... while ((nBytes = jdeFread(szChunk, sizeof(ZCHAR), nSize, inFile)) != 0) { ... } I081500_MD5_Final(lpBhvrCom, lpVoid, lpDS, (BYTE *)szDigest, &mdContext); IntToMathNumeric (nFileByteSize, &lpDS- >mnFileByteSize); ... </pre>	<p>The jdeFread function parameter 3 uses jdesizeV_t (another name for size_t data type). The variable being passed in is of data type int (32-bit signed)</p> <p>In 32-bit builds: jdesizeV_t (that is size_t) is 32-bit unsigned. int is 32 signed.</p> <p>Even though the sign is changing, the variable width is the same. This is an acceptable conversion (with our current compiler settings). No warnings are given.</p> <p>In 64-bit builds: jdesizeV_t (that is size_t) is 64-bit unsigned. int is 32 signed.</p> <p>Here, you have a change of both sign and width of the variable value. This promotion will throw the warning that you see on AIX. You can make the warning go away by changing the value passed to jdeFread from signed to unsigned. Then the unsigned int will automatically promote from 32-bit unsigned to 64-bit unsigned.</p> <p>Note that this is only acceptable if the actual valid range of values is from 0 to INT_MAX (+2 GB).</p>	<pre> int nSize = 0; size_t uSize = 0; ... MathNumericToInt(&lpDS- >mnFileChunkSize, &nSize); ... uSize = nSize; while ((nBytes = jdeFread(szChunk, sizeof(ZCHAR), uSize, inFile)) != 0) { ... } I081500_MD5_Final(lpBhvrCom, lpVoid, lpDS, (BYTE *)szDigest, &mdContext); IntToMathNumeric(nFileByteSize, &lpDS- >mnFileByteSize); ... </pre>	
-----------------------	-------	--	---	--	---	--

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
				<p>Recommendations: Try to change the nSize data type to size_t or unsigned int. Many times, this will solve several problems at once.</p> <p>In this specific example, int nsize is passed to several other functions that also require int data type, so this change is not practical. In this case, creating a temporary variable is an easier Recommendation.</p>		

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
time_t issue	AIX64	1506-742 (I) 64-bit portability: possible loss of digits through conversion of long int type into unsigned int type.	<pre> /* Seed the Random Generator to Current Time */ srand((unsigned long int) time((time_t *)NULL)); ... iRandom = jdePPRand(); </pre>	<p>srand is a function provided by the C standard library. Input value is unsigned int.</p> <p>time() is a function, also provided by the C standard library, that returns a signed long.</p> <p>In 32-bit builds: unsigned int is 32-bit unsigned. time_t is 32 signed.</p> <p>Even though the sign is changing, the variable width is the same. This is an acceptable conversion (with our current compiler settings). No warnings are given.</p> <p>In 64-bit builds: unsigned int is 32-bit unsigned. time_t is 64-bit signed.</p> <p>Here, a 64-bit value will be truncated. The code then calls jdePPRand() to retrieve the random number.</p> <p>jdePPSRand() and jdePPRand() are meant to work as a team. JDE jdePPSRand also supports the use of jdePPSrand(time(NULL)), which is common practice.</p>	<pre> /* Seed the Random Generator to Current Time */ jdePPSrand(time(NULL)); ... iRandom = jdePPRand(); </pre>	

Integer type mismatch	AIX64 HP64 WIN64	<p>AIX64 1506-742 (I) 64-bit portability: possible loss of digits through conversion of long int type into int type. HP64 warning #4229-D: 64-bit migration: conversion from "off_t" to "int" may truncate value WIN64 warning C4242: 'function' : conversion from '_int64' to 'jdesize_t', possible loss of data</p>	<pre> long nFileLen = 0; struct jdeZStat info; " jdeStat(pSourcePath, &info); nFileLen = info.st_size; if(nFileLen == -1) { ... } pBuffer = (BYTE*) jdeAlloc(COMMON_POOL, nFileLen, MEM_ZEROINIT); jdeFread(pBuffer, sizeof(BYTE), nFileLen, inputFp); </pre>	<p>The first issue is the value width. info.st_size is an off_t type which will be signed 64-bit integer in 64-bit builds. The code converter will change nFileLen from long to int. The result is an assignment that will truncate the statement nFileLen = info.st_size;</p> <p>A second issue is the difference in signage. info.st_size is signed. However, other uses of nFileLen are to pass it to jdeFRead and jdeFWrite where 64-bit unsigned (jdesizeV_t) values are all expected. nFileLen = info.st_size;</p> <p>HP and AIX will throw an error if you try to change both the width and signage of a value. 32-bit option: NOTE: This Recommendation is only appropriate if the file that is being read contains less than 2GB of data. If you change just the signage (that is, if you change long nFileLen, to unsigned int), you still have a problem with the line "if (nFileLen == -1)". This can be overcome by delaying the transition of the value from signed to unsigned. You use a temporary int variable to evaluate the results of iFileLen = jdeCast32(info.st_size) and a jdeCast32() to downsize the result of info.st_size. Then, after the -1 value is considered, you reassign the value to an unsigned int variable</p>	<pre> int iFileLen = 0; unsigned int nFileLen = 0; struct jdeZStat info; " jdeStat(pSourcePath, &info); iFileLen = jdeCast32(info.st_size); if(nFileLen == -1) { ... } nFileLen = iFileLen; pBuffer = (BYTE*) jdeAlloc(COMMON_POOL, nFileLen, MEM_ZEROINIT); jdeFread(pBuffer, sizeof(BYTE), nFileLen, inputFp); </pre>	<pre> off_t nFileLen = 0; struct jdeZStat info; " jdeStat(pSourcePath, &info); nFileLen = info.st_size; if(nFileLen == -1) { ... } pBuffer = (BYTE*) jdeAllocV(COMMON_POOL, nFileLen, MEM_ZEROINIT); jdeFread(pBuffer, sizeof(BYTE), nFileLen, inputFp); </pre>
-----------------------	------------------------	---	--	--	---	--

Class	Platform	Platform Message	Sample Code	Discussion	Apps 9.2 Forward Compatible Fix	Apps 64-Bit Fix (If Different)
				<p>where it can be auto-promoted in parm 3 of jdeFRead or jdeFWrite. (Note that jdeAlloc accepts a 32-bit unsigned int in parm 2 and the unsigned int works here).</p> <p>The second alternative is to continue using the data type off_t for nFileLen; the same data type returned by the info.st_size_t value. Long will not work because it is renamed by the code converter. You can continue using off_t for nFileLen. Then, you need to use the 64-bit jdeAllocV() to allocate memory.</p>		

ORACLE CORPORATION

Worldwide Headquarters

500 Oracle Parkway, Redwood Shores, CA 94065 USA

Worldwide Inquiries

TELE + 1.650.506.7000 + 1.800.ORACLE1

FAX + 1.650.506.7200

oracle.com

CONNECT WITH US

Call +1.800.ORACLE1 or visit oracle.com. Outside North America, find your local office at oracle.com/contact.



Integrated Cloud Applications & Platform Services

Copyright © 2020, Oracle and/or its affiliates. All rights reserved. This document is provided for information purposes only, and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document, and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group. 0620

Business Brief Title

January 2017

Author: [OPTIONAL]

Contributing Authors: [OPTIONAL]

