LIBDDM
PACK FUNCTION
EXTENDED FORMAT
BUGFIX

S. de Graaf

Circuits and Systems Group
Faculty of Electrical Engineering
Delft University of Technology
The Netherlands

Report ET-CAS 99-05
November 18, 1999

Last revision: December 16, 2003.

1. INTRODUCTION

This report describes a bug fix made to the libddm _dmPack() function. This bug can happen when writing doubles with the F format to the NELSIS database in dm_extended_format. The dm_extended_format is a new way for writing doubles with two D formats to the database. This 32 bits integers has a bigger range then the old F format. In the old F format, doubles are written as one scaled integer using a multiplication factor of 1000.

This work is carried out for the SPACE project on 2 November 1999.

2. USAGE OF THE F FORMAT

For the usage of the F format, see also the "dmpdata.c" source file.

This format is only used by two GEOMETRY (layout) data files, for storage of non-orthogonal layout elements:

(1) The "nor" stream (using GEO_NOR_INI and GEO_NOR_XY records).
(2) The "nxx" stream (using GEO_NXX_INI and GEO_NXX_XY records).

2.1 The NOR Stream

The NOR stream is used for storage of non-orthogonal layout features. Layout storage programs, like *dali* and *cldm*, are using it.

The data of a non-orthogonal layout element is written using a GEO_NOR_INI record and a number of GEO_NOR_XY records. The GEO_NOR_INI record specifies the layer and the element type, the number of GEO_NOR_XY records used and the element bounding boxes. The GEO_NOR_INI record specifies also the number of x- and y-copies used and there 'dx' and 'dy' values. Note that this 'dx' and 'dy' values are written with the F format.

The GEO_NOR_XY records specify the coördinate points of the element type. This 'x' and 'y' values are written with the F format to the database.

2.2 The NXX Stream

The NXX stream is used for storage of expanded non-orthogonal layout features. The layout expansion program *makeboxl* is writing this format.

The data of a non-orthogonal layout element is written using a GEO_NXX_INI record and a number of GEO_NXX_XY records. The GEO_NXX_INI record specifies the element type and the number of GEO_NXX_XY records used.

In case the element type is a CIRCLE_NOR, the GEO_NXX_INI record also contains six additional F format values (xc, yc, r1, r2, a1, a2).

The GEO_NXX_XY records specify the coördinate points of the element type. This 'x' and 'y' values are written with the F format to the database.

3. THE OLD F FORMAT

In the old F format, the double is written as a scaled integer. To gain more resolution, the double is first multiplied with 1000 and after that rounded to an integer. That implies that the value of the double may not be bigger than MAXINT / 1000, else an integer overflow error occurs. For a 32 bits integer (32 = 4 x 8 bits):

```
MAXINT = 0x7fffffff =  2,147,483,647
MININT = 0x10000000 = -2,147,483,648
```

Because

```
maxint = 0x001fffff (= 2,097,151)
```

is used, the double ’d’ must be in the range:

```
-2,097,151.0 <= d <= 2,097,151.0
```

A read double ’d’ is converted into a integer ’n’ with the following statement:

```
n = (long) (1000 * d + (d > 0 ? 0.5 : -0.5));
```

The mantissa is rounded to three digits behind the comma.
The value of ’n’ is in the range:

```
-2,097,151,000 <= n <= 2,097,151,000
```

This integer value is written in the D format to the database.

4. THE NEW F FORMAT

The new F format, also called the extended format, stores a double as two integers. One integer for the part before the comma and one integer for the part after the comma (the mantissa). This allows a better range and accuracy. We now use:

```
maxint = 0x7fffffff (= 2,147,483,647)
```

The double 'd' must be in the new range:

```
-2,147,483,647.0 <= d <= 2,147,483,647.0
```

Double values:

```
[-] n9 n8 ... n2 n1 n0 '.' [0...] m9 m8 ... m2 m1 m0
```

are written with the LSD first and leading zeros after the mantissa:

```
[-] n0 n1 ... n9 sep m0 m1 ... m9 [0...]
```

The following code is used to calculate 'n' and 'm':

```
n = (long)(d + 0.5e-15 * d); /* rounding also ok if d < 0 */
e = d > 0 ? d - n : n - d;

prec = 15;
for (tmp = d < 0? -d : d; tmp >= 1.0; tmp *= 0.1) --prec;
sprintf (buf, %.*lf, prec, e);
for (i = 0; buf[i] != '.'; i++);
z = 0;
for (j = i+1; buf[j] == '0'; j++) leading_zeros++;
if (buf[j] && e > 0) {
    z = j - i;
    while (buf[++j]) { if (buf[j] != '0') z = j - i; }
    fac = 10.0;
    while (--z) fac *= 10.0;
    while ((tmp = e * fac + 0.5) >= maxdouble + 1.0) fac *= 0.1;
    m = (long)tmp;
}
else m = 0;
```

Note that the maximum value of '0.5e-15 * d' is:

```
0.5e-15 * d <= 0.0000010737418235
```

The double 'd' is truncated to the integer 'n'. Why some rounding must be applied is unknown. In most cases is n < d for d > 0 and n > d for d < 0. When 'n' is the number before the comma, then is 'e' the fraction (mantissa).

NOTE: When 'e' is < 0, then the value of 'n' is rounded to a new value.
In that case the mantissa needs not to be written!

After that, there is done a lot of calculations to make the mantissa 'm'. The fraction is maximum 15 digits behind the comma. When the absolute value of d is ≥ 1, then the fraction is 14. When d ≥ 10, then the fraction is 13. And so on. Because the maximum

value of d ≡ 2147483647, the value of ’prec’ is always ≥ 5. The buffer ’buf’ contains the mantissa, but it can contain leading and trailing zero’s. The leading zero’s are counted and ’z’ is the count of the not zero digits after the leading zero’s. This digits is the value of ’m’ and may not be > 2147483647. If you do not write the mantissa with the D format, you can handle it as ASCII characters and simple write them, but you must also read them as ASCII characters back.

For example:

```
i = prec;
while (buf[i]) ++i;
while (buf[--i] == ’0’);
if (buf[i] != ’.’) {
    do {
       n = buf[i] - ’0’;
       PUTNIBBLE (n);
    }
    while (buf[--i] != ’.’);
}
else m = 0;
```

4.1 Discussion of Changes in the Source Files

For a list of changes see appendix C.

4.1.1 First change

For negative ’d’ values the precision ’prec’ must be calculated like positive ’d’ values:

```
< for (tmp = d; tmp > 0.999999999999999; tmp *= 0.1)
> for (tmp = d < 0? -d : d; tmp > 0.999999999999999; tmp *= 0.1)
```

4.1.2 Second change

The mantissa may not be > maxint:

```
>  while ((tmp = e * fac + 0.5) >= maxdouble + 1.0) fac *= 0.1;
>  m = (long)tmp;
```

5. APPENDICES

APPENDIX A -- SCCS Delta's

FILE:	PREVIOUS:		NEW:	
name	bytes	delta/date	bytes	delta/date
dmpack.c	16550	3.25 99/03/31	16593	3.26 99/11/02

APPENDIX B -- Test Program

```
% cat max.c

main ()
{
   int i, n = 1, m = 0;
   double d;
   /*
   for (i = 1; i <= 32; ++i) {
      m+= n;
      printf("i = %2d, n = %d, m = %d (%x), m1=%d\n", i, n, m, m, m+1);
      n<<= 1;
   }
   */
   n = 0x7fffffff;
   printf ("n = %d (%x)\n", n, n);
   d = n;
   printf ("d = %32.20f\n", d);
   d *= 0.5e-15;
   printf ("d = %32.20f\n", d);
}
```

APPENDIX C -- Changes

```
===================================================================
< static char *SccsId = "@(#)dmpack.c 3.25 (TU-Delft) 03/31/99";
> static char *SccsId = "@(#)dmpack.c 3.26 (TU-Delft) 11/02/99";
===================================================================
17,18d16
< Modification date : 26-Apr-1988
> Modification date : 02-Nov-1999
196c196
<     for (tmp = d; tmp > 0.999999999999999; tmp = tmp * 0.1)
>     for (tmp = d < 0? -d : d; tmp > 0.999999999999999; tmp = tmp * 0.1)
219c219
<     for (j = i+1; buf[j] == '0' && buf[j] != '\0'; j++)
>     for (j = i+1; buf[j] == '0'; j++)
221,224c221,234
<     if (buf[j] == '\0') leading_zeros = 0;
<
<     for (j = i; buf[j] != '\0'; j++) {
<         if (buf[j] != '0') z = j - i;
---
>     if (buf[j] && e > 0) {
>         z = j - i;
>         while (buf[++j]) {
>            if (buf[j] != '0') z = j - i;
>         }
>         fac = 10.0;
>         while (--z) fac *= 10.0;
>         /*
>         ** Because e always larger or equal to zero, we can use
>         ** only one expression for avoiding truncation errors.
>         */
>         while ((tmp = e * fac + 0.5) >= maxdouble + 1.0) fac *= 0.1;
>         m = (long)tmp;
226,229c236,239
<     fac = 1.0;
<     while (z > 0) {
<         fac = fac * 10.0;
<         z--;
---
>     else {
>         leading_zeros = 0;
>         fac = 1.0;
>         m = 0;
231,236d240
<     /*
<     ** Because e always larger or equal to zero, we can use
<     ** only one expression for avoiding truncation errors.
<     */
<     m = (long)(e * fac + 0.5);
```