Precision and some other problems in ENDF-6 format: thoughts of a programmer.

V.Zerkin, IAEA, Nuclear Data Section, 01-09 November 2012

Problems of ENDF-6 format mentioned in presentations and private communications (M.Herman, D.Brown, R.Capote) on several recent nuclear data meetings can be summarized as following:

1) Rigid structure not allowing extensions
2) Fixed number of digits for data presentation (not enough precision for covariance data)
3) Dropped 'E' in REAL numbers (allowed by FORTRAN, but not by other languages)
4) Impossible to read by a human
5) Punch-card structure of information (repeating MAT/MF/MT)
6) Short MAT (4 digits only)
7) Short MT (3 digits)[, and long procedure to add new official MT]
8) Extensive usage of MT=5 for many reactions/products
9) Complex coding needed to read, no standard software support
10) [Also mentioned: price to rewrite large codes dealing with ENDF-6 format in case it would be replaced by another format/s or modern language/s.]
11) etc.

Part-1. Precision of floating point numbers in ENDF-6 format: realistic improvement.

Basic data line in ENDF-6 file stores 6 floating point numbers (11 positions each) and four integer numbers: MAT (i4), MF (i2), MT (i3) and NS (i5). What is the possible maximum accuracy of the real numbers in ENDF-6 basic data line which can be read by standard FORTRAN READ statement? ENDF-6 Manual declares precision on the levels of 5, 6 and 7 digits, but theoretically correct answer is 11. Of course, the answer 11 is valid only for positive values between 1e+10 and (1e+11 - 0.5), and in practice answer depends of the values of data.

Single and double real floating point numbers in IEEE standard have values in the range shown in the table below, though usually we assume that mantissa has precision 7 and 15 digits and exponent has limits ±37 and ±307.

Range:	from	To
REAL*4 (single precision)	**±1.17549435e-38**	**±3.40282347e+38**
REAL*8 (double precision)	**±2.2250738585072013d-308**	**±1.7976931348623158d+308**

Having width =11 symbols, normally, we can present most of our data with precision 5 digits (mantissa) with 2 digits assigned in exponent: [**±.12345±E12**]. Program can write data to ENDF-6 file using simple FORTRAN statement:

```
write (out,'(6e11.5,i4,i2,i3,i5)') (data(i),i=n+1,n+6),mat,mf,mt,ns
```

More advanced programs improve accuracy of data in output file using special algorithms:

```
implicit double precision (a-h,o-z)
character*11 out11 ! function: converts double -> character*11 with max precision
write (out,'(a11,i4,i2,i3,i5)') (out11(data(i)),i=n+1,n+6),mat,mf,mt,ns
```

Reading programs should be able to input data by simple FORTRAN statement READ to read formatted text data (today FORTRAN compilers are very flexible reading numbers from text files):

```
implicit double precision (a-h,o-z)
do ...
  read (in,'(6e11.0,i4,i2,i3,i5)') (data(i),i=n+1,n+6),mat,mf,mt,ns
enddo
```

How the accuracy of data stored in output ENDF-6 file can be improved in practice? For example, FORTRAN can read formatted data with fixed width without symbol 'E' for exponent (it should have sign explicitly written) - this is widely used in ENDF files. Also, for some values, data can be stored without exponent at all - written using format 'F' instead of 'E' (partially implemented in PREPRO-2007). So, having in a program REAL*8 variable R=123.456789123456 instead of writing to the file [1.2346E+02] we can write [123.4567891] improving accuracy from 5 to 10 digits (this is correct only for positive numbers values between 0.1 and 9999999999.5).

Are all these tricks well known? Are all possible improvements already fully implemented? Not true - even recently released libraries have ENDF-6 files with data which can be presented with better accuracy in many-many cases. How? In three ways: (a) by eliminating space before positive numbers, (b) by dropping decimal point, (c) wide usage of 'F' format. For example,

(a) instead of [**1.767138-5**] we can have [**1.7671381-5**] improving accuracy from 7 to 8 digits*
(b) instead of [**-7.31592-10**] we can have [**-7315921-16**] improving accuracy from 6 to 7 digits*
(ab) instead of [**4.68006-10**] we can have [**4680061 2-17**] improving accuracy from 6 to 8 digits*
(c) instead of [**-1.269414+2**] we can have [**-126.941412**] improving accuracy from 7 to 9 digits**

**Values from original ENDF-6 file of JENDL-4.0u library (release 2012), U-233, MF33, MT1*
***Value from original ENDF-6 file of ENDF-B/VII.1, Mn-55, MF32, MT151*

Examples of presentation of REAL*8 variables in ENDF-6 files with precision better than 7 digits (can be read by usual FORTRAN formatted reading statements):

In memory:	*File:123456789.1*	*Mantissa*	*Exponent*
1.23456789123D-038	12345679-45	*8 digits*	*2 digits*
1.23456789123D+106	12345679+99	*8 digits*	*2 digits*
-1.23456789123D-093	-1234568-99	*7 digits*	*2 digits*
1.23456789123D+017	123456789+9	*9 digits*	*1 digit*
-1.23456789123D-002	-12345679-9	*8 digits*	*1 digit*
.123456789123	.1234567891	*10 digits*	*none*
123.456789123	123.4567891	*10 digits*	*none*
1234567891.23	1234567891.	*10 digits*	*none*
-123.456789123	-123.456789	*9 digits*	*none*
12345678912.3	12345678912	*11 digits*	*none*
99999999999.3	99999999999	*11 digits*	*none*
-1234567891.23	-1234567891	*10 digits*	*none*
-9999999999.23	-9999999999	*10 digits*	*none*

```
 25-Mn- 55 IAEA Eval-Feb11 IAEA
MF33:MT16                    +digits
[ 1.795040-1] -> [.1795040123] +3
[ 8.454340-2] -> [.0845434012] +2
[ 9.532870-3] -> [95328701-10] +1
MF32:MT151
[ 4.145117-3] -> [41451171-10] +1
[ 1.120613-2] -> [.0112061312] +2
[-1.108435+0] -> [-1.10843512] +2
[-7.367702-5] -> [-7367702-11] ==
[-1.081405-3] -> [-.001081405] ==
[-6.542516-4] -> [-6542516-10] ==
[-3.285303-2] -> [-32853031-9] +1
[-1.269414+2] -> [-126.941412] +2
[ 9.687644+1] -> [96.87644123] +3
[ 4.772108+3] -> [4772.108123] +3
[-7.530070-5] -> [-7530070-11] ==
 [-7.02505-17] -> [-7025051-23] +1
```

Improvement of precision in ENDF-6 file to have additionally 1, 2, 3 digits in mantissa may look unimportant, but if it goes via chain of programs it can accumulate some difference in the final results – see in Appendix-1 part of PENDF file produced by PREPRO codes with advanced conversion and "standard" conversion developed in PREPRO - even number of intervals was changed, because energy was stored more precisely.

Conclusion

Precision of ENDF-6 files can be improved by 1 to 3 digits (from 6-7 to 7-10) with small modifications of writing part of programs and without any modifications of reading part.

Part-2. Removing limitations in data precision (and capacity for integers) in ENDF-6 files: unrealistic solutions, hiding format, software interface.

Part-1 of this document describes ways of improvement of precision of the data stored in ENDF-6 files without any modifications of the reading part of the programs. Part-2 describes options which can significantly improve precision – these are radical solutions which would not only change format, but also require revision of both input and output parts of the programs dealing with ENDF-6 files. (Most likely they are rather theoretical and never be implemented.)

General remark.

The main problem in any significant change in ENDF-6 format is not really a change of data format – it is the necessity to modify a lot of programs dealing with ENDF-6 files (with subsequent testing and debugging). This is obvious. What is not obvious – how many really big/complex programs should be modified and how complex are these modifications. The most important question is: how good (or whether at all) is input/output (I/O) part of the programs separated from the main logic of program? In another words: if read/write statements are mixed with calculations then program source should be modified in many places with high risk of mistakes; if program reads all necessary data from ENDF-6 file to internal data structures (variables and arrays), then performs calculations and outputs results if necessary to ENDF-6 file, then such a program can be modified easier.

Solution-1. "ENDF-6*2" (doubled ENDF-6) (wide ENDF-6)

Extension of the width (precision of real numbers and capacity of integer MAT, MF, MT, NS) could be done, for example, by simply doubling the space for every number (i.e. we go from 80 columns to 160). Stupid? May be. But if such modification would be in accepted for cases where precision is really crucial (as some physicists claim), modification of programs would be pretty simple, straight forward and does not need any specific knowledge and intelligence efforts – just change everywhere in reading formats from '(6E11.0,I4,I2,I3,I5)' to '(6E22.0,I8,I4,I6,I10)' for real data and MAT/MF/MT/NS for the basic lines, and trivial changes for MF1/MT451 section.

Solution-2. "ENDF-6*" (ENDF-6 star)

Another solution: to stop using fixed width of the numbers (real and integer) in the basic lines and read all numbers using format (*). Variable width will allow any precision and will remove limitations to MAT numbers and numbers of MT (number of reactions). This would need somewhat more complex modifications of existing ENDF-6 files: now blanks will not be allowed inside one number; blank fields have to be properly recognized and replaced by zero's. Some changes have to be done also for MF1/MT451 section. Reading statements in FORTRAN should be changed
from

```
READ (in,'(6E11.0, I4,I2,I3,I5)') (data(i),i=1,6),mat,mf,mt,ns
```

to

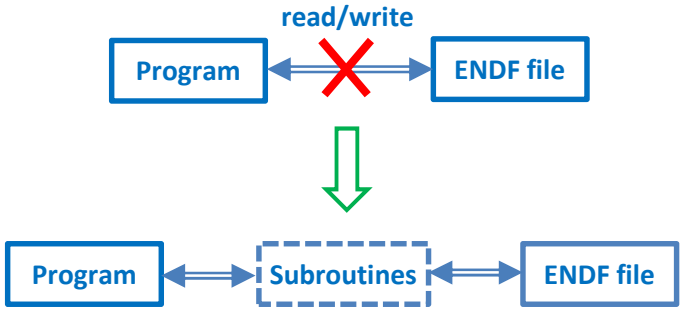
```
READ (in,*) (data(i),i=1,6),mat,mf,mt,ns
```

or we can move all READ statements to common subroutine and call it

```
CALL endf6_read1line(in,data,mat,mf,mt,ns)
```

So, in this type of subroutines we can hide actual implementation of input (and format!) from the main program, for example, we can change width from 11 to 22 or use variable width and format (*), drop NS from the file and generate it on the fly, etc.

This is an important point: we have to enforce programs to read/write data to the files only via subroutines in order to prepare future changes in format.



Solution-3. "ENDF - 6+" (ENDF-6 plus)

If we agree to modify programs reading and writing ENDF data files and allow this access only via software interface, we can think about more radical changes in ENDF files.

Why do we keep MAT/MF/MT/NS column on the right side* and repeat them in every line? This is one of the factors limiting data precision. It is usually ignored in the reading of data arrays. We can have it only in a Record Headers. Do we need NS in every line? For eye checking (?); but having limit =5 digits, when it becomes greater than 99999 it restarts from 1, and it is misleading.

**Note: EXFOR files, EXFOR dictionaries, X4TOC4 dictionaries are also designed with such a fashion, and there are also problems with growing length of the reaction-string, limited space for description of data columns, etc.*

Thinking even further about possible modifications of ENDF format in order to make it more flexible, allowing universal programming and future extension of data structures, we can come to the need to introduce there explicit description of ENDF Structures and Records - i.e. basic data containers. This means having in the file records with text "SECT", "TEXT", "CONT", "HEAD", "DIR", "LIST", "TAB1", "TAB2" with parameters describing nested data structures (and finally arrays of numbers) located below these descriptive records. Using these descriptors we could automatically recognize low level structures and therefore should be able to write universal subroutines implementing input/output for low level data containers. We could also (perhaps) build interface for top level blocks - section or even for full evaluation. This would be very deep change, and it has to be compared with alternative options - main of them is format using XML language.

Conclusion

Solutions 1 and 2 look like patches and do not solve real problems of rigid structure, and do not open the gates for future extensions, but the idea of separation of programs from I/O implementation and accessing data only via subroutines can be useful for future extensions.

Solution-3 can solve many problems of ENDF-6 format, it can be very effective for FORTRAN programs; comparing to XML - it does not have standard supporting software and tools, less universal, and needs more programming, but it can be simpler, faster and independent from external software producers.

Part-3. XML language for storage of evaluated nuclear data files: steps of implementation.

Using XML language can really solve the most of rigid structural problems of ENDF6 format. It is really universal and modern solution directed to the future. Although comprehensive implementation of all internal structures of evaluated nuclear data in XML language needs much effort, one of the most difficult problems will be the adaptation of existing big/complex codes to use new format. And there are two parts of this problem:

1) to prepare software interface for existing programs accessing data in the new format. In practice this would mean making a FORTRAN library of accessing ENDF files and structures
2) to adopt existing (mainly FORTRAN) codes to use this interface. This would mean deep inspection and modification of the source codes and may be even the structure of the programs

So, we have two large tasks: to develop a new format based on XML and implement smooth transition of existing programs to use of new format. Note: although many individuals have developed their own I/O software libraries (on C, Java, FORTRAN languages) dealing with ENDF format, until now there is no common ENDF FORTRAN library. If such a FORTRAN library (even

simple I/O library) existed and would be used by big codes, then we would not really have major problems switching to XML.

Can we start to develop such a library for dealing with XML nuclear data files? No! XML for evaluated nuclear data is under development itself; it is not yet established and probably will be modified many times; we cannot really develop library with the format changing; we cannot guarantee that it will be fully working and debugged.

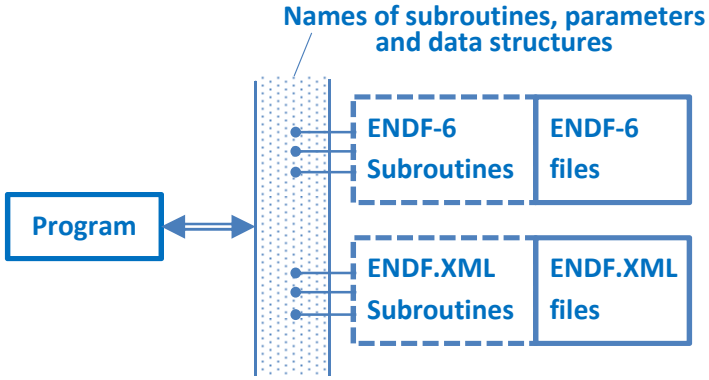
How we can separate these two tasks and work on them in parallel and overall optimize our efforts and final result?

Step-1. Because ENDF-6 format is well established and big codes work with ENDF-6 structured files (using ENDF data structures and logic), we can start to develop FORTRAN I/O library for work with ENDF-6 format (or select one of existing libraries). This task of step-1: to develop ENDF I/O FORTRAN library. It is not simple task. But even only analysis of requirements, planning data structures, subroutines on the top level and creating detailed specification of such library, without complete implementation, would be very useful for further steps.

Step-2. Adopting big/complex/important codes to use this library. Separate input/output from the main logic of the codes.

> **XML.** Development and implementation of XML format for evaluated nuclear data and tools will go in parallel with steps 1 and 2.

Step-3. When (and if) XML for ENDF will be ready and accepted, this FORTRAN I/O library has to be replaced by I/O interface to XML without changes in existing big codes.



Conclusion

Although it may look strange to start development of a universal FORTRAN library for ENDF-6 files after 20 years of existence of ENDF-6 format, and at the time when nuclear data community begins the development of a new (XML) format to replace ENDF-6 format, nevertheless, the steps described above can help to solve some current problems of ENDF-6 format, to prepare software structure to make transition to XML format for existing codes: smooth, understandable, tested and confirmed at the end.

```
92233.00000231.0380000          0          0          0          09222 3  1    1 92233.0000 231.038000          0          0          0          09222 3  1    1
0.00000E+000.00000E+00          0          0          1      37865 9222 3  1    2 0.0        0.0                 0          0          1      37841 9222 3  1    2
      37865          2                                          9222 3  1    3      37841          2                                          9222 3  1    3
10000000-1229138.47642110303691227744.31508111082001227653.38846 9222 3  1    4 1.00000E-5 29138.4764 1.10304E-5 27744.3151 1.11082E-5 27653.4218 9222 3  1    4
12166905-1226416.86957123385581226243.77064134205461225152.94807 9222 3  1    5 1.21669E-5 26416.8696 1.23386E-5 26243.7177 1.34205E-5 25152.9482 9222 3  1    5
13705900-1224904.55961148033581223949.51122152240021223633.45357 9222 3  1    6 1.37059E-5 24904.5354 1.48034E-5 23949.5111 1.52240E-5 23633.4880 9222 3  1    6
16328651-1222803.66535169111001222425.90732180111051221712.65507 9222 3  1    7 1.63287E-5 22803.6653 1.69111E-5 22425.9269 1.80111E-5 21712.6550 9222 3  1    7
18784224-1221279.83658198669151220673.85698208659001220191.10692 9222 3  1    8 1.87842E-5 21279.8448 1.98669E-5 20673.8570 2.08659E-5 20191.0935 9222 3  1    8
21913941-1219684.77300231770241219157.90892241718871218743.02489 9222 3  1    9 2.19139E-5 19684.7730 2.31770E-5 19157.9144 2.41719E-5 18743.0248 9222 3  1    9
25745500-1218176.43598266624841217846.34778285971061217245.07834 9222 3  1   10 2.57455E-5 18176.4413 2.66625E-5 17846.3477 2.85971E-5 17245.0806 9222 3  1   10
29409705-1216992.58566317662001216360.40342324399911216179.68522 9222 3  1   11 2.94097E-5 16992.5857 3.17662E-5 16360.4050 3.24400E-5 16179.6852 9222 3  1   11
35284705-1215520.94114357825091215405.69169391949001214723.60598 9222 3  1   12 3.52847E-5 15520.9408 3.57825E-5 15405.6917 3.91949E-5 14723.6003 9222 3  1   12
39469430-1214668.74371435362391213967.06894480220811213298.99596 9222 3  1   13 3.94694E-5 14668.7437 4.35362E-5 13967.0689 4.80221E-5 13298.9960 9222 3  1   13
48699586-1213211.87937529701291212662.89795544752991212495.84074 9222 3  1   14 4.86996E-5 13211.8792 5.29701E-5 12662.8980 5.44753E-5 12495.8380 9222 3  1   14
58428010-1212057.24532609360051211816.97645644482541211480.58213 9222 3  1   15 5.84280E-5 12057.2454 6.09360E-5 11816.9783 6.44483E-5 11480.5822 9222 3  1   15
68162943-1211173.40809710888051210931.52160762469871210563.35049 9222 3  1   16 6.81629E-5 11173.4131 7.10888E-5 10931.5216 7.62470E-5 10563.3505 9222 3  1   16
78413579-1210408.74452858914001209948.04724686493075129910.987723 9222 3  1   17 7.84136E-5 10408.7445 8.58914E-5 9948.04877 8.64931E-5 9910.98772 9222 3  1   17
95405058-129437.073517967510511299375.27653110523530-1118985.851149 9222 3  1   18 9.54051E-5 9437.07352 9.67511E-5 9375.27406 1.05235E-4 8985.85117 9222 3  1   18
10898900-118837.098694116078431188556.22913412363500-1118297.760927 9222 3  1   19 1.08989E-4 8837.08504 1.16078E-4 8556.22915 1.23635E-4 8297.75984 9222 3  1   19
12803880-118147.168831140249001177786.68859114123152-1117757.688692 9222 3  1   20 1.28039E-4 8147.16883 1.40249E-4 7786.70213 1.41232E-4 7757.68868 9222 3  1   20
15578359-117386.873311160210251177289.48710217183505-1117033.809518 9222 3  1   21 1.55784E-4 7386.87330 1.60210E-4 7289.49882 1.71835E-4 7033.80952 9222 3  1   21
18302100-116821.406133189540411166697.64354020907008-1116377.569101 9222 3  1   22 1.83021E-4 6821.40087 1.89540E-4 6697.64355 2.09070E-4 6377.56910 9222 3  1   22
21054500-116356.699739230612021166072.83907924390411-115910.511486 9222 3  1   23 2.10545E-4 6356.69866 2.30612E-4 6072.83907 2.43904E-4 5910.51559 9222 3  1   23
25437358-115782.687214280583461155506.42440628454199-115470.402299 9222 3  1   24 2.54374E-4 5782.68721 2.80583E-4 5506.42441 2.84542E-4 5470.39870 9222 3  1   24
30949392-115243.404123334292001155048.64186134138323-114992.961539 9222 3  1   25 3.09494E-4 5243.40412 3.34292E-4 5048.64060 3.41383E-4 4992.96154 9222 3  1   25
37655832-114754.523481839826251114627.50949241535774-114527.492928 9222 3  1   26 3.76558E-4 4754.52348 3.98263E-4 4627.50666 4.15358E-4 4527.49293 9222 3  1   26
45815494-114311.335490481192001144210.88630550536183-114105.524844 9222 3  1   27 4.58155E-4 4311.33549 4.81192E-4 4210.88680 5.05362E-4 4105.52485 9222 3  1   27
55743277-113909.568837589562251133805.18633861486894-113722.990342 9222 3  1   28 5.57433E-4 3909.56884 5.89562E-4 3805.18759 6.14869E-4 3722.99034 9222 3  1   28
67822316-113545.350002737697621133401.38632174810520-113376.199798 9222 3  1   29 6.78223E-4 3545.35000 7.37698E-4 3401.38493 7.48105E-4 3376.19979 9222 3  1   29
82518768-113215.164784910212501133061.82275494928295-113000.992375 9222 3  1   30 8.25188E-4 3215.16479 9.10213E-4 3061.82276 9.49283E-4 3000.99275 9222 3  1   30
10039980-102915.825963110744691032776.82039712215549-102644.455794 9222 3  1   31 .001003998 2915.82596 .001107447 2776.82039 .001221555 2644.45579 9222 3  1   31
12651280-102600.821422134742021032518.43466714862542-102398.446684 9222 3  1   32 .001265128 2600.82142 .001347420 2518.43467 .001486254 2398.44669 9222 3  1   32
16393933-102284.191995177075101032199.58748018083114-102175.401491 9222 3  1   33 .001639393 2284.19199 .001770751 2199.58723 .001808311 2175.40149 9222 3  1   33
19946343-102071.830892220015531031973.21157724268526-101879.307993 9222 3  1   34 .001994634 2071.83089 .002200155 1973.21157 .002426853 1879.30799 9222 3  1   34
26396672-101803.213032267690811031789.90014629527286-101704.793851 9222 3  1   35 .002639667 1803.21310 .002676908 1789.90015 .002952729 1704.79385 9222 3  1   35
32569687-101623.770458359255681031546.63618539627229-101473.200430 9222 3  1   36 .003256969 1623.77046 .003592557 1546.63619 .003962723 1473.20043 9222 3  1   36
43406250-101408.479784437102981031403.27377848214074-101336.686198 9222 3  1   37 .004340625 1408.47981 .004371030 1403.27377 .004821407 1336.68620 9222 3  1   37
53181905-101273.251780586616071031212.82493964705920-101155.282140 9222 3  1   38 .005318191 1273.25178 .005866161 1212.82494 .006470592 1155.28214 9222 3  1   38
71373020-101100.509912787270781031048.36476083852130-101016.985263 9222 3  1   39 .007137302 1100.50991 .007872708 1048.36476 .008385213 1016.98528 9222 3  1   39
86838876-10998.702530095786489-1031951.3833394.0105656037906.2899886 9222 3  1   40 .008683888 998.702530 .009578649 951.383339 .010565604 906.289988 9222 3  1   40
.011654251386 3.3386120.012855069822.4510281.014179617078 3.5519836 9222 3  1   41 .011654251 863.338612 .012855070 822.451028 .014179617 783.551984 9222 3  1   41
.015640641574 6.5500615.017252205571 1.3049023.019029820167 7.6847093 9222 3  1   42 .015640641 746.550061 .017252205 711.304902 .019029820 677.684709 9222 3  1   42
.020990594764 5.6324068.022672479062 1.8980662.023153401661 5.1125256 9222 3  1   43 .020990595 645.632406 .022672479 621.898071 .023153402 615.112525 9222 3  1   43
.025241880058 9.4610509.025300000058 8.7930558.027906834856 1.0083139 9222 3  1   44 .025241880 589.461050 .025300000 588.793055 .027906835 561.008314 9222 3  1   44
.030782270053 4.5283937.033953981250 9.2976546.037452495848 5.2659909 9222 3  1   45 .030782270 534.528393 .033953981 509.297654 .037452496 485.265991 9222 3  1   45
.041311486846 2.3764723.045568096444 0.5730738.050263294041 9.8027147 9222 3  1   46 .041311487 462.376473 .045568096 440.573074 .050263294 419.802714 9222 3  1   46
.055442270540 0.0153708.061154873038 1.1642905.067456084636 3.2065960 9222 3  1   47 .055442271 400.015370 .061154873 381.164290 .067456085 363.206596 9222 3  1   47
```

PENDF file – result of PREPRO illustrating effect of my subroutine R8toChar11.for in comparison with OUT9X.F (conversion from REAL*8 to CHARACTER*11)