Subject: C/FORTRAN/IDL/(MATLAB) XDR routines - SOURCE (part-1??)
Posted by dutch on Mon, 22 Mar 1993 11:38:11 GMT
View Forum Message <> Reply to Message

--
Here are (some of) the C/Fortran/IDL/(MATLAB) routines
I promised everyone so long ago! The routines were written by a
colleague (Jean-Marc Moret) and I, for our own purposes, to
allow easier data exchange/sharing between IDL/MATLAB/FORTRAN etc.
I am posting them here in the belief that they may be of use to others.
Unfortunately, I have been forced to omit most of the
MATLAB-specific routines for the time being, due to filename clashes
between the C routines and MEX user_fcn files - I am at the mercy of
my colleague in this regard. I will try include these in a second
posting (to complete the MATLAB interface), at a later date.
The routines which I have assembled here allow data to be
exchanged between C/FORTRAN/IDL on various machines. The files will
have to be extracted manually from this listing, so will require a
small effort - my apologies.

DISCLAIMER: We are physicists, not computer professionals, so
the routines may not be as elegant as possible. We have used the
routines for some time here and they satisfy our requirements.
I hope you find the routines useful too. If you have any
suggestions (i.e. improvements or bug reports) I would be glad
to hear them, but cannot promise to undertake further development!

Summary of files in this listing:

1) XDRLIB.C collection of C routines
which run on VAX/VMS , SUN/Unix and Cray
The routines can be called from Fortran
as shown in the examples, or linked to form
MATLAB MEX files (some work required here, or wait for
second posting)
NOTE: the C buffer routines which are called from FORTRAN
are lowercase on VAX and SUN machines, but MUST BE
UPPERCASE ON THE CRAY! (If the routine names are not UPPERCASE,
the link will fail)
2) FORTRAN sample routines (+ command files)
to produce XDR save files using the routines in XDRLIB
3) IDL procedures to LOAD/SAVE XDR files.
4) MAT_FORMAT.DOC Briefly describes the
MATLAB file format, which we have used for our XDR files
The MATLAB format was chosen for simplicity (i.e. XDR
files can be simply converted to .MAT MATLAB files or
vice-versa, as all the necesary information is in
the data files).

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%% C:  XDRLIB.C %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
/*
=======================================
 miscellaneous XDR I/O routines
 for VAX/VMS and SUN.
 Jean-Marc Moret + Michael Dutch
         Sep 1992
 tested on SUN: Sep 1992
 tested on VAX: Feb 1993
=======================================
*/

# ifdef VMS
# include "MULTINET_ROOT:[MULTINET.INCLUDE.RPC]RPC.H"
# include stdio
# include string
# else
# include <memory.h>
# include <malloc.h>
# include <stdio.h>
# include <string.h>
# include <rpc/rpc.h>
# endif

FILE *fp;
XDR xdrs;

/*
=======================================
 routine to open an XDR file
 callable from Fortran
 Michael Dutch  Sep 1992
=======================================
*/

/* int XDROPEN(fname,flen) ######## This Declaration is for the Cray */
int xdropen(fname,flen)
char *fname;
long *flen;
{
/* printf("xdropen: filename,len=%s  %i \n",fname,flen); */

 if (strlen(fname) == 0)
    {printf("xdropen: no filename specified ???\n");return -2;};
```

```
 /* open the file and link it to the XDR stream */
 if ((fp = fopen(fname,"w")) == NULL)
    {printf("xdropen: problem opening the file.\n");return -1;};
        xdrstdio_create(&xdrs, fp, XDR_ENCODE);
 return 0;
}
/*
=======================================
 routine to close an XDR file
 callable from Fortran
 Michael Dutch  Sep 1992
=======================================
*/
/* int XDRCLOSE() ####### This Declaration is for the Cray */
int xdrclose()
{
 /* close the file */
 if (fclose(fp))
    {printf("xdrclose: problem closing file.\n");return -1;}
 return 0;
}
/*
=====================================================
   XDRSAVE: Fortran to C buffer routine (M.J.Dutch)
   closely modelled on J-M. Moret savexdr MEX file.
   calls J-M. Moret's savexdr.c to do the hard work!
 ===tested on VAX and SUN so far===
   variable names are passed as INT*4 arrays to
   avoid complications with string arguments.
=====================================================
*/
/* int XDRSAVE(type,name,m,n,imagf,preal,pimag,namlen) ###### CRAY */
int xdrsave(type,name,m,n,imagf,preal,pimag,namlen)
int *type, *m, *n, *imagf, *namlen;
char *name, *preal, *pimag;
{
int status;

/* DEBUGGING */
/*
printf("\n type: %u \n",*type);
printf(" name: %s \n",name);
printf("    m: %u \n",*m);
printf("    n: %u \n",*n);
printf("imagf: %u \n",*imagf);
printf("preal: %p \n",preal);
printf("pimag: %p \n",pimag);
*/
```

```
/* save the variable */

status=savexdr(&xdrs, *type, name, *m, *n, *imagf, preal, pimag);
if (status != 0) printf("savexdr: error writing to file.\n");
return status;

}
/*
    ================================================
 Enhanced load and save routines for MAT-files
    ================================================

         ) Jean-Marc Moret Software

    TCV - Tokamak ` configuration variable
 Centre de Recherches en Physique des Plasmas
   Icole Polytechnique Fidirale de Lausanne
       CH-1015 Lausanne,  Switzerland
*/

# ifdef VMS
# if CC$gfloat == 1
# define MATMACHINETYPE 3000
# else
# define MATMACHINETYPE 2000
# endif
# else
# define MATMACHINETYPE 0
# endif

# include "matfile.h"

/*----------------------------------------*/

typedef struct {
      long type;
      long mrows;
      long ncols;
 long imagf;
 long namlen;
} Fmatrix;

int  sizeofmattype();
xdrproc_t xdr_mattype();

/* savexdr
 *********/
```

```
int  savexdr(xdrs,type,pname,mrows,ncols,imagf,preal,pimag)

/* return -1 in case of write failure
        -2 in case of memory problem
        -3 with invalid type */

XDR  *xdrs;
int  type;
int  mrows;
int  ncols;
int  imagf;
char  *pname;
char  *preal;
char  *pimag;
{
 Fmatrix  M;
 int  mn, elmtSize;
 xdrproc_t xdr_filter;
 register int i;

 M.type   = type;
 M.mrows  = mrows;
 M.ncols  = ncols;
 M.imagf  = imagf;
 M.namlen = strlen(pname) + 1;
 mn = M.mrows * M.ncols;

 /* estimate element size and set xdr filter */
 if (!(elmtSize = sizeofmattype(M.type))) return (-3);
 xdr_filter = xdr_mattype(M.type);

 /* write matrix header, the namlen is written by xdr_bytes with the name */
 if (xdr_long(xdrs, &M.type)   == FALSE ||
     xdr_long(xdrs, &M.mrows)  == FALSE ||
     xdr_long(xdrs, &M.ncols)  == FALSE ||
     xdr_long(xdrs, &M.imagf)  == FALSE)
  return (-1);

 /* write matrix name */
 if (xdr_bytes(xdrs, &pname, &M.namlen, MAXMATRIXNAMESIZE) == FALSE)
  return (-1);

 /* write real part */
 for (i = 0; i < mn; i++)
  if (xdr_filter(xdrs, preal+i*elmtSize) == FALSE)
   return(-1);
```

```
 /* write imag part */
 if (imagf)
  for (i = 0; i < mn; i++)
   if (xdr_filter(xdrs, pimag+i*elmtSize) == FALSE)
     return(-1);

 /* done */
 return (0);
}

 /*-------------------------------------------------------- -----*/

/* loadxdr
 *********/
int  loadxdr(xdrs, type, pname, mrows, ncols, imagf, preal, pimag)

/* return -1 in case of read failure
         -2 in case of memory problem
         -3 with invalid type */

XDR   *xdrs;
int   *type;
int   *mrows;
int   *ncols;
int   *imagf;
char   *pname;
char   **preal;
char  **pimag;

{
 Fmatrix  M;
 int  mn, elmtSize;
 xdrproc_t xdr_filter;
 register int i;

 /* get Fmatrix structure from file, the namlen will be read by xdr_bytes */
 if (xdr_long(xdrs, &M.type)   == FALSE ||
     xdr_long(xdrs, &M.mrows)  == FALSE ||
     xdr_long(xdrs, &M.ncols)  == FALSE ||
     xdr_long(xdrs, &M.imagf)  == FALSE)
  return(-1);
 *type  = M.type;
 *mrows = M.mrows;
 *ncols = M.ncols;
 *imagf = M.imagf;
 mn = M.mrows * M.ncols;
 /* estimate element size */
 if (!(elmtSize = sizeofmattype(M.type))) return (-3);
```

```
 xdr_filter = xdr_mattype(M.type);

 /* allocate memory */
 if (!(*preal = malloc(mn*elmtSize)))
  return(-2);
 if (M.imagf)
  if (!(*pimag = malloc(mn*elmtSize))) {
   free(*preal);
   return(-2);
  }

 /* get matrix name */
 if (xdr_bytes(xdrs, &pname, &M.namlen, MAXMATRIXNAMESIZE) == FALSE) {
  free(*preal); free(*pimag);
  return (-1);
 }
 /* get real part */
 for (i = 0; i < mn; i++)
  if (xdr_filter(xdrs, (*preal)+i*elmtSize) == FALSE)
   return(-1);

 /* get imag part */
 if (M.imagf)
  for (i = 0; i < mn; i++)
   if (xdr_filter(xdrs, (*pimag)+i*elmtSize) == FALSE)
     return(-1);

 /* done */
 return(0);
}

 /*-------------------------------------------------------- -----*/

/* a function to get the element size, returns a null size in case of invalid format */
int  sizeofmattype(type)

int  type;

{
 switch(type % 100 - type % 10){
  case MATDOUBLE:
   return(sizeof(double));
  case MATFLOAT:
   return(sizeof(float));
  case MATLONG:
   return(sizeof(long));
  case MATSHORT:
   return(sizeof(short));
```

```
  case MATUSHORT:
   return(sizeof(unsigned short));
  default:
   return(0);
 }
}

 /*---------------------------------------------------- ---*/

# ifdef VAX
# if CC$gfloat == 1
extern xdrproc_t xdr_gfloat();
# endif
# endif

/* a function to get the xdr filter function returns a null pointer in case of invalid format */
xdrproc_t xdr_mattype(type)

int  type;

{
 switch(type % 100 - type % 10){
  case MATDOUBLE:
# ifdef VAX
#  if CC$gfloat == 1
   return(xdr_gfloat);
#  else
   return(xdr_double);
#  endif
# else
   return(xdr_double);
# endif
  case MATFLOAT:
   return(xdr_float);
  case MATLONG:
   return(xdr_long);
  case MATSHORT:
   return(xdr_short);
  case MATUSHORT:
   return(xdr_u_short);
  default:
   return(NULL);
 }
}

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%% C: MATFILE.H include file %%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
# define MATDOUBLE 0
# define MATFLOAT 10
# define MATLONG 20
# define MATSHORT 30
# define MATUSHORT 40
# define ISREAL 0
# define ISCOMPLEX 1
# define COLUMNWISE 0
# define ROWWISE 100
# define ISTEXT 1
# define ISNOTTEXT 0
# define MAXMATRIXNAMESIZE 64

# ifdef VMS
# if CC$gfloat == 1
# define MATMACHINETYPE 3000
# else
# define MATMACHINETYPE 2000
# endif
# else
# define MATMACHINETYPE 0
# endif

int  loadmat(), savemat();
int  loadxdr(), savexdr();

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%% FORTRAN:  TESTXDR_VAX.FOR %%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c Fortran test routine to call XDR output
c routines written in C.
c *** VAX/VMS Version ***
c   Michael Dutch, Feb 1993
c
 program testxdr
 external nullterm
 character*30 nullterm,testfile
 integer m,n,imagf,xint
 integer status,xdropen,xdrclose,xdrsave
 integer*4 intarr(30)
 integer*4 inam(30),xnam(30),ynam(30),cnam(30),snam(30)
 real xreal,ximag
 real*8 yreal,yimag
 dimension xint(2,3)
 dimension xreal(2,3),ximag(2,3)
 dimension yreal(2,3),yimag(2,3)
 character*30 iname,cname,xname,yname
 character*30 sname,strng
```

```
 data testfile/'testxdr_vax.xdr'/
 data iname/'intmat'/,cname/'cplxmat'/
 data xname/'realmat'/,yname/'dblemat'/
 data sname/'strng'/,strng/'Hello World'/
c
 print*,'Enter the filename: '
 read(*,'(A)') testfile

 m=2
 n=3
 imagf=0
 do 10 i=1,2
 do 20 j=1,3
   xint(i,j)=3*(i-1)+j
   xreal(i,j)=float(3*(i-1)+j)
   ximag(i,j)=float(3*(i-1)+j)
   yreal(i,j)=dble(3*(i-1)+j)
   yimag(i,j)=dble(3*(i-1)+j)
20 continue
10 continue

c---store strings as INT*4 arrays (for simpler parameter passing)---
 do 30 i=1,30
    intarr(i)=ichar(strng(i:i))
30 continue

c++++++++++++save the variables++++++++++++++++++++++++++
 print*,'before xdropen'
 status=xdropen(%REF(nullterm(testfile)))
 print*,'after xdropen, status= ',status
 if (status.ne.0) print*,'xdropen failed, status= ',status
  status=xdrsave(20,%REF(nullterm(iname)),m,n,imagf,xint,xint)
 print*,'after xdrsave,status= ',status
 if (status.ne.0) print*,'xdropen failed, status= ',status
  status=xdrsave(10,%REF(nullterm(xname)),m,n,imagf,xreal,xima g)
  status=xdrsave(0,%REF(nullterm(yname)),m,n,imagf,yreal,yimag )
 status=xdrsave(10,%REF(nullterm(cname)),m,n,1,xreal,ximag)
 status=xdrsave(21,%REF(nullterm(sname)),1,index(strng,'  '),imagf,
     +   intarr,intarr)
 status=xdrclose()
 if (status.ne.0) print*,'xdrclose failed, status= ',status
c++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 end

 character*(*) function nullterm(str)
c --adds null termination to string--
 character*(*) str
 n=index(str,' ')
```

```
 if (n.eq.0) n=len(str)
 nullterm(1:n)=str(1:n-1)//char(0)
 return
 end
```

%%%
%%%%%%
%%%%%%%%%%%%% DCL: LINKXDR_VAX.COM %%%%%%%%%%%%%%%%%%%%%%%
%%%
%%%%%%

```
$! VAX/VMS command file to link XDR routines
$! fort testxdr_vax
$! cc xdrlib.c
$
$ LINK testxdr_vax,xdrlib,sys$share:vaxcrtl/lib, -
      MULTINET_ROOT:[MULTINET.LIBRARY]RPC.OLB/LIBRARY
```

%%%
%%%%%%
%%%%%%%%%%%%% FORTRAN: TESTXDR_SUN.F %%%%%%%%%%%%%%%%%%%%%
%%%
%%%%%%

```
c test routine to call SAVEXDR routines
c written in C, from Fortran
c   Michael Dutch, SEP 1992
c
 program testxdr
 external xdropen  !$pragma C (xdropen)
 external xdrsave  !$pragma C (xdrsave)
 external xdrclose !$pragma C (xdrclose)
 character*30 testfile,nullterm
 integer*4 m,n,imagf,xint,intarr
 integer status,xdropen,xdrclose,xdrsave
 real xreal,ximag
 real*8 yreal,yimag
 dimension xint(2,3),intarr(30)
 dimension xreal(2,3),ximag(2,3)
 dimension yreal(2,3),yimag(2,3)
 character*30 iname,cname,xname,yname
 character*30 sname,strng
 data testfile/'xdrfile.xdr'/
 data iname/'intmat'/,cname/'cplxmat'/
 data xname/'realmat'/,yname/'dblemat'/
 data sname/'strng'/,strng/'Hello World'/
c
 print*,'Enter the filename: '
 read(*,'(A)') testfile
```

```
 m=2
 n=3
 imagf=0
 do 10 i=1,2
 do 20 j=1,3
   xint(i,j)=3*(i-1)+j
   xreal(i,j)=float(3*(i-1)+j)
   ximag(i,j)=float(3*(i-1)+j)
   yreal(i,j)=dble(3*(i-1)+j)
   yimag(i,j)=dble(3*(i-1)+j)
20 continue
10 continue

c---convert string to INT array---
 do 30 i=1,30
    intarr(i)=ichar(strng(i:i))
30 continue

c+++++++++++save the variables+++++++++++++++++++++++
 status=xdropen(nullterm(testfile))
 status=xdrsave(20,nullterm(iname),m,n,imagf,xint,xint)
 status=xdrsave(10,nullterm(xname),m,n,imagf,xreal,ximag)
 status=xdrsave(0,nullterm(yname),m,n,imagf,yreal,yimag)
 status=xdrsave(10,nullterm(cname),m,n,1,xreal,ximag)
 status=xdrsave(21,nullterm(sname),1,30,imagf,intarr,intarr)
 integ=42
 realv=1.23
 status=xdrsave(20,nullterm('integ'),1,1,imagf,integ,integ)
 status=xdrsave(10,nullterm('realv'),1,1,imagf,realv,realv)
 status=xdrclose()
c++++++++++++++++++++++++++++++++++++++++++++++++++++
 end

 character*(*) function nullterm(str)
c --adds null termination to string--
 character*(*) str
 n=index(str,' ')
 if (n.eq.0) n=len(str)
 nullterm(1:n)=str(1:n-1)//char(0)
 return
 end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%%%%%%%%%%%%%%%%%%%%%% LINKXDR_SUN %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
# --- SUN commmand file to link xdr routines ---
```

```
#! /bin/csh
# f77 testxdr_sun.f
# cc -c xdrlib.c
f77 -o testxdr testxdr_sun.o xdrlib.o

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%% IDL: LOADXDR.PRO %%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;+
;
;  PURPOSE:  loads an XDR file created by MATLAB, IDL etc
;            'file' must contain the name of the file to be loaded
;
;  USAGE:    file='filename'
;            @loadxdr
;
;  NOTES:    1) repeatedly calls 'loadxdr_var.pro' to load
;               a single variable, until EOF is reached.
;            2) '.run user:[dutch.public]loadxdr_var' may be
;               necessary before @loadxdr (e.g. in startup.pro)
;
;  AUTHOR:   M.J.DUTCH    JAN-92
;  MODIFS:   M.J.DUTCH  1-MAR-93    Avoid using EOF which does not
;        work over DECNET.
;
;-
xdr_unit=1
close,xdr_unit
openr,/xdr,xdr_unit,file
xdr_stat=0

while (xdr_stat ne 2) do begin $
  loadxdr_var,xdr_unit,xdr_x,xdr_name,xdr_stat & $
;** help,xdr_x,xdr_name & $
  if xdr_stat eq 0 then xdr_z=execute(xdr_name+'=xdr_x')

close,xdr_unit
;**** the following clean_up line only works at the main level!! ****
delvar,xdr_unit,xdr_x,xdr_name,xdr_z,xdr_stat

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
%%%%%%%%%%%%%%%% IDL: LOADXDR_VAR.PRO %%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
PRO loadxdr_var,unit,x,name,status
;+
;
```

```
;  PURPOSE:   loads one variable from the specified unit
;        called repeatedly from 'loadxdr' to load all variables
;   M.J.Dutch Jan-1992
;
;  USAGE:     loadxdr_var,unit,var,name,status
;
;  ARGUMENTS: unit   = logical unit number  (input)
;       var    = the variable read from unit (output)
;       name   = name of the variable  (output)
;       status = status flag   (output)
;  0=normal return, 1=empty variable, 2=end_of_file
;
;  NOTES:     If a variable name conflicts with an IDL reserved name
;             the variable name is modified.
;
;  AUTHOR:    M. J. DUTCH    JAN-92
;  MODIFS:    M. J. DUTCH  1-MAR-93 Avoid using EOF which does not
;     work over DECNET.
;
;-
ON_IOERROR,io_error
vartypes=['DOUBLE',' FLOAT',' LONG ','  INT ',' BYTE ']
 reserved=['AND','BEGIN','CASE','COMMON','DO','ELSE','END','E NDCASE', $
   'ENDELSE','ENDFOR','ENDIF','ENDREP','ENDREPEAT','ENDWHI',  $
   'ENDWHILE','EQ','FOR','FUNCTION','GE','GOTO','GT','IF',    $
   'LE','LT','MOD','NE','NOT','OF','ON_IOERROR','OR','PRO',   $
   'REPEAT','THEN','UNTIL','WHILE','XOR']
header=lonarr(4)
arrlen=long(0)
type=long(0)
status=0

readu,unit,header
;**print,'header= ',header

type=header(0)
if type lt 0 then goto,finish
type=type-100*fix(type/100) ;set 100's & 1000's to zero
prec=fix(type/10)
text=type-10*prec
;**print,'type,prec,text=',type,prec,text

mrows=header(1)
ncols=header(2)
imagf=header(3)

;--check whether it is an empty matrix--
if mrows*ncols eq 0 then begin
```

```
   status=1
   goto,finish
endif

bytname=bytarr(30)
readu,unit,bytname
ind=where(bytname > 0)
if ind(0) eq -1 then name='unnamed' else name=string(bytname(ind))
;**print,'name,prec,imagf=',name,prec,imagf

 ;*-------------------------DEBUGGING---------------------- ----------
;*if (imagf eq 0) then begin
;*   if (text eq 0) then begin
;*      print,STRCOMPRESS('reading a '+STRING(mrows)+' by '+ $
;*      STRING(ncols)+' REAL '+vartypes(prec)+' array named '+name)
;*   endif else begin
;*      print,STRCOMPRESS('reading a '+STRING(mrows)+' by '+ $
;*      STRING(ncols)+' TEXT '+vartypes(prec)+' array named '+name)
;*   endelse
;*endif else begin
;*   print,STRCOMPRESS('reading a '+STRING(mrows)+' by '+STRING(ncols)+ $
;*   ' COMPLEX '+vartypes(prec)+' array named '+name)
;*endelse
 ;*------------------------------------------------------- ---------

case prec of
;----DOUBLE----
0:begin
  if imagf eq 0 then begin
     x=dblarr(mrows,ncols)
     readu,unit,x
     if text eq 1 then begin
        if (mrows eq 1) then x=string(transpose(byte(x))) $
        else x=transpose(string(byte(transpose(x))))
     endif
  endif else begin
     areal=dblarr(mrows,ncols)
     aimag=dblarr(mrows,ncols)
     readu,unit,areal
     readu,unit,aimag
     x=complex(areal,aimag)
  endelse
  end
;----FLOAT----
1:begin
  if imagf eq 0 then begin
     x=fltarr(mrows,ncols)
     readu,unit,x
```

```
    if text eq 1 then begin
      if (mrows eq 1) then x=string(transpose(byte(x))) $
      else x=transpose(string(byte(transpose(x))))
    endif
  endif else begin
    areal=fltarr(mrows,ncols)
    aimag=fltarr(mrows,ncols)
    readu,unit,areal
    readu,unit,aimag
    x=complex(areal,aimag)
  endelse
  end
;----LONG----
2:begin
  if imagf eq 0 then begin
    x=lonarr(mrows,ncols)
    readu,unit,x
    if text eq 1 then begin
      if (mrows eq 1) then x=string(transpose(byte(x))) $
      else x=transpose(string(byte(transpose(x))))
    endif
  endif else begin
    areal=lonarr(mrows,ncols)
    aimag=lonarr(mrows,ncols)
    readu,unit,areal
    readu,unit,aimag
    x=complex(areal,aimag)
  endelse
  end
;----SHORT----
3:begin
  if imagf eq 0 then begin
    x=intarr(mrows,ncols)
    readu,unit,x
    if text eq 1 then begin
      if (mrows eq 1) then x=string(transpose(byte(x))) $
      else x=transpose(string(byte(transpose(x))))
    endif
  endif else begin
    areal=intarr(mrows,ncols)
    aimag=intarr(mrows,ncols)
    readu,unit,areal
    readu,unit,aimag
    x=complex(areal,aimag)
  endelse
  end
;----BYTE----
4:begin
```

```
  if imagf eq 0 then begin
     x=bytarr(mrows,ncols)
     readu,unit,x
     if text eq 1 then begin
;**     print,'restoring byte array to string'
        if (mrows eq 1) then x=string(transpose(byte(x))) $
        else x=transpose(string(byte(transpose(x))))
     endif
  endif else begin
     areal=bytarr(mrows,ncols)
     aimag=bytarr(mrows,ncols)
     readu,unit,areal
     readu,unit,aimag
     x=complex(areal,aimag)
  endelse
  end
else:print,'** unrecognised data type **'
endcase

;CHECK FOR CONFLICTS WITH IDL RESERVED NAMES
;**print,(strupcase(name) eq reserved)
if total(strupcase(name) eq reserved) GT 0 then begin
   print,'LOADXDR: Variable name conflicts with IDL reserved name'
   print,'         ',name,' will be renamed ','_'+name
   name='_' + name
endif

finish:
return

io_error:
status=2
goto,finish
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%%%%%%%%%%%%% IDL: SAVEXDR.PRO %%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
PRO savexdr,file,var,name,append
;+
;
;  PURPOSE:   saves one variable in the specified file
;             using XDR (eXternal Data Representation) format
;             must be called repeatedly to save all variables
;
;  USAGE:     savexdr,file,var,name,append
```

```
;
;  ARGUMENTS: file = output filename in which to save data (string)
;       var  = variable to be saved
;       name = name under which variable is saved (string)
;       append = 'c' to create new file
;         'a' to append to an existing file
;
;  AUTHOR:    M. J. DUTCH   JANUARY 1992
;
;-
;-----------------------------------------------
;**help,file,var,name,append
;-----------------------------------------------
typeconv=[-1,4,3,2,1,0]

if n_params() ne 4 then begin
   print,'savexdr requires 4 parameters ?'
   goto,finish
endif

append=strupcase(strtrim(append,2))
if append ne 'A' and append ne 'C' then begin
   print,'append must be either "A" or "C" ?'
   goto,finish
endif
get_lun,unit
if append eq 'A' then openw,/append,/xdr,unit,file $
                else openw,/xdr,unit,file

;if variable is text then convert to integer array
;(so it can be read back by MATLAB) and set text flag
text=0
sizevect=size(var)
idltype=sizevect(sizevect(0)+1)
if idltype eq 7 then begin
;**   print,'converting text strings to integer arrays'
   text=1
   xold=var
   var=transpose(reform(fix(byte(var))))
endif

;now get variable size and type
sizevect=size(var)
ndim=sizevect(0)
idltype=sizevect(ndim+1)
mrows=1
ncols=1
if ndim ge 1 then mrows=sizevect(1)
```

```
if ndim ge 2 then ncols=sizevect(2)
if ndim ge 3 then begin
   print,'variable has too many dimensions ?'
   goto,finish
endif
;** print,'idltype=',idltype

;separate real and imaginary components
imagf=0
if idltype eq 6  then begin
   imagpart=float(imaginary(var))
   var=float(var)
   imagf=1
   type=1
endif else begin
   type=typeconv(idltype)
endelse

type=10*type + text

;set up the header
header=long([type,mrows,ncols,imagf])

;save the data
writeu,unit,header
writeu,unit,[byte(name),byte(0)] ;store name as bytes, so we can add
    ;correct (null-terminated) length
writeu,unit,var
if imagf eq 1 then begin
   writeu,unit,imagpart
endif

finish:
if text eq 1 then var=xold ;restore text strings
free_lun,unit
return
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%%%%%%%%%%%%%%%%% MAT_FORMAT.DOC %%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
---- SUMMARY OF MATLAB FILE FORMAT ----
(Painfully typed excerpts from the MATLAB manual)
FOR USERS OF IDL LOADXDR/SAVEXDR, WHO ARE NOT
FAMILIAR WITH THE MATLAB FILE FORMAT.
 Michael DUTCH, 16-MAR-1993
```

The save command saves MATLAB variables on disk in a specially structured file we call a MAT (XDR) file...

A MAT (XDR) file may contain one or more variables. The variables are written sequentially on disk, with the bytes conceptually forming a continuous stream. Each variable starts with a fixed 20-byte header that contains information describing certain attributes of the variable. The 20-byte header consists of 5 four-byte long integers (words):

TYPE Type flag. Word 1 contains an integer whose decimal digits encode the variable type. If the integer is represented as MOPT where M is the thousands digit, O the hundreds digit, P the tens digit, and T is the ones digit, then:

M indicates the numeric format of binary numbers on the machine that wrote the file. Use this table to determine number to use for your machine: PC 0
Other Intel 0
Sun 1
Apollo 1
Macintosh 1
Other Motorola 1
VAX D-float 2
VAX G-float 3
NOTE THAT THIS VARIABLE IS NOT USED IN THE IDL LOADXDR/SAVEXDR ROUTINES AND IS IMPLICITLY ASSUMED TO BE ZERO!

O in normally 0 (zero), which the data are stored in a column-wise orientation (varies fastest down a column). If O=1, the data are transposed and stored in a row-wise orientation (varies fastest across a row).
NOTE THAT THE IDL ROUTINE SAVEXDR USES ONLY O=0 (i.e. same as MATLAB) AND THE IDL ROUTINE LOADXDR ALSO ASSUMES O=0.

P is normally zero (for .MAT files, NOT .XDR files - MJD), which means that the data are stored on disk in double-precision (8 bytes/ element). If P=1, the data are stored in single precision (4 bytes/ element). P=2 is signed 32-bit integer data, P=3 is 16-bit signed integers, and P=4 is unsigned 16-bit integers.

T is normally zero, indicating thta the data that fiollow describe a matrix. If T=1, the variable is a text variable. This means that the numbers in the variable are floating point numbers between 0 and 255 representing the ASCII code of characters.

For PCs, type is usually 0000, or 0, which indicates PC double precision matrix data stored by columns. Note that P.ne.0 and O=1

are not produced by the (MATLAB) SAVE command (THIS IS NOT TRUE FOR THE IDL SAVEXDR ROUTINE - MJD), but could be generated outside of MATLAB (to save file space) and are accepted by (MATLAB) LOAD (IDL LOADXDR TOO! - MJD).

[e.g. in the IDL routine SAVEXDR, TYPE=20 implies integer*4 data, whilst TYPE=31 implies a text string stored as short integers (integer*2).]

MROWS Row dimension. Word 2 contains an integer with the row dimension of the variable.

NCOLS Column dimension. Word 3 contains an integer with the column dimension of the variable.

IMAGF Imaginary flag. Word 4 is an integer that is either 0 or 1. If 1, then the variable has an imaginary part. If 0, there is only real data.

NAMLEN Name length. Word 5 contains an integer with the length of the variable name plus one.

Immediately following the fixed length header is the data that has a length dependent on the variables in the fixed length header:

NAME Variable name. The name consists of NAMLEN ASCII bytes, the last one of which must be a NUL character (encoded as 0).

REAL Real part of the matrix. The real data consist of MROW*NCOL double precision (8-byte) floating point numbers (or as otherwise specified by TYPE). Matrices are stored column-wise, first the first column, then the second column, etc, unless otherwise specified by TYPE in the fixed header.

IMAG Imaginary part of the matrix, if any. If the imaginary flag IMAGF is nonzero, the imaginary part of a matrix is here. It is stored in the same way as REAL data.

The structure is repeated for each variable stored in the file.

%%
%%%%%%%%%%%%%%%%%% THAT'S ALL FOLKS ! %%%%%%%%%%%%%%%%%%
%%

##

Michael Dutch   email: dutch@elpp1.epfl.ch

Centre de Recherches en Physique des Plasmas
Ecole Polytechnique Federale de Lausanne
21 Ave des Bains Aussie.Abroad
CH-1007 Lausanne, SWITZERLAND _--_|\
#---------- / \ #
I'd rather have a full bottle in front _.--._/
of me than a full frontal lobotomy. v #
##########