# SHL and MDS Packages

**Bill Meyer**

**Corsica Winter School – ASIPP**

**January 26-28, 2016**

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Security, LLC, Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

# Listing Packages in Corsica

```
corsica> list packages
Priority Name           Long Name                    Status
                                 ...
    17 mds      mds: MDSPlus access routines        -- up --
    15 shl      SHL: Dynamic linking loader rout    -- up –
                                 …
corsica> list shl.functions
call_external()
   builtin        function

exec_external()
   builtin        function

close_library(libpath)
   integer        function
```

# SHL – Shared Library Package

- Corsica/Basis interface to dynamic linking loader (dll) routines dlopen, dlsym, and dlclose.
  - dll is what reads .so libraries via LD_LIBRARY_PATH
  - Supports run-time execution of shared objects/libraries
    - Just as IDL supports shared library external code (call_external)
  - Allows users to execute compiled foreign code without requiring Corsica to be rebuilt.
    - Binary only distributions
  - README and C and Fortran examples in repository doc/package_shl folder.

- All communication passed through arguments.
  - No direct access to Corsica/Basis DB.
- Shared libraries must be complete with all entry points resolved.

# SHL – Function call_external

- **Recognizable to IDL developers**
  - **call_external(<str shared lib>, <str entry point>, [arg1], .....)**
  - **Easy port from IDL to Corsica**
  - **Generally requires a C "wrapper" code since arguments passed as a count and array of addresses rather than individual arguments**
    - **Better control over entry point name**
  - **bar.c example in docs/package_shl**

```
int bar(argc,argv)
int argc;       /*      number of arguments   */
void *argv[];   /*      array of pointers to the arguments   */
{

        int i;
        float *a;       /*      1st argument    */
        int *len;       /*      2nd argument  */
```

# SHL – Bar.c continued

```
        if(argc != 2){
            printf("usage: status = bar(<shared library>,\"bar\",<array>,<len>)\n");
              return 1;
        }
        a = (float *)argv[0];
        len = (int *)argv[1];

        for(i = 0; i < *len; ++i)a[i] = (float) i;
        return 0;
}
```

- **Compile code for shared object**
  - **Position independent**
    - **Compiler dependent but usually with -fpic**

```
gcc bar.c -fpic -c
ld -share bar.o -o libfoo.so
```

# SHL – Shared library function bar execution

- Memory must be allocated by basis
- {real} overrides Corsica macro that makes all reals double precision by default.
  - Variable types within Corsica and compiled code must match
- Basis makes copies of variables, “&” passes original.
- Be wary of array indice base.
  - Array base 1 in Corsica is base 0 in C and base 1 in Fortran

```
corsica> {real} foo(10)
corsica> integer status = call_external("./libfoo.so","bar",&foo,10)
corsica> foo
foo              shape: (10)
 (1)          0.00000E+00   1.00000E+00   2.00000E+00   3.00000E+00
 (5)          4.00000E+00   5.00000E+00   6.00000E+00   7.00000E+00
 (9)          8.00000E+00   9.00000E+00
```

# SHL – Function exec_external

- **exec_external(<str shared lib>, <str entry point>, [arg1], .....)**
- **Subroutine called with argument list**
- **fbar.f example in docs/package_shl**

```
integer function fbar(a,len)
 real*4 a(*)
 integer len,i
 do i=1,len
        a(i) = i
 enddo
 fbar = len
 return
 end
```

```
pgf90 -fpic -c fbar.f
ld -shared fbar.o -o libfoo.so
ldd -d libfoo.so
        statically linked
undefined symbol: pgf90_compiled        (./libfoo.so)
```

# Shared Libraries Must Resolve All Symbols

```
pgf90 -shared fbar.o -o libfoo.so
ldd -d libfoo.so
        linux-gate.so.1 =>  (0xffffe000)
        libc.so.6 => /lib/tls/libc.so.6 (0x40018000)
        libpgc.so => /afs/fepcluster/usr/pgi/grendel/5.2/lib/libpgc.so (0x40133000)
        libm.so.6 => /lib/tls/libm.so.6 (0x40147000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x80000000)
```

```
corsica> {real} foo(10)
corsica> exec_external("./libfoo.so","fbar_",&foo,10)
exec_external("./libfoo.so","fbar_",&foo,10) =   10
corsica> foo
foo               shape: (10)
 (1)         1.00000E+00   2.00000E+00   3.00000E+00   4.00000E+00
 (5)         5.00000E+00   6.00000E+00   7.00000E+00   8.00000E+00
 (9)         9.00000E+00   1.00000E+01
```

# SHL Summary

- Last function - close_library(<str library path>)
  - Close all references to specified library
  - Required to map new version or reset library global storage
- SHL has been used to call into shared libraries developed for IDL to analyze data from DIII-D
  - DIII-D uses IDL extensively and has many existing libraries that could be called from Corsica
- Fairly new package
  - No active applications that we know of

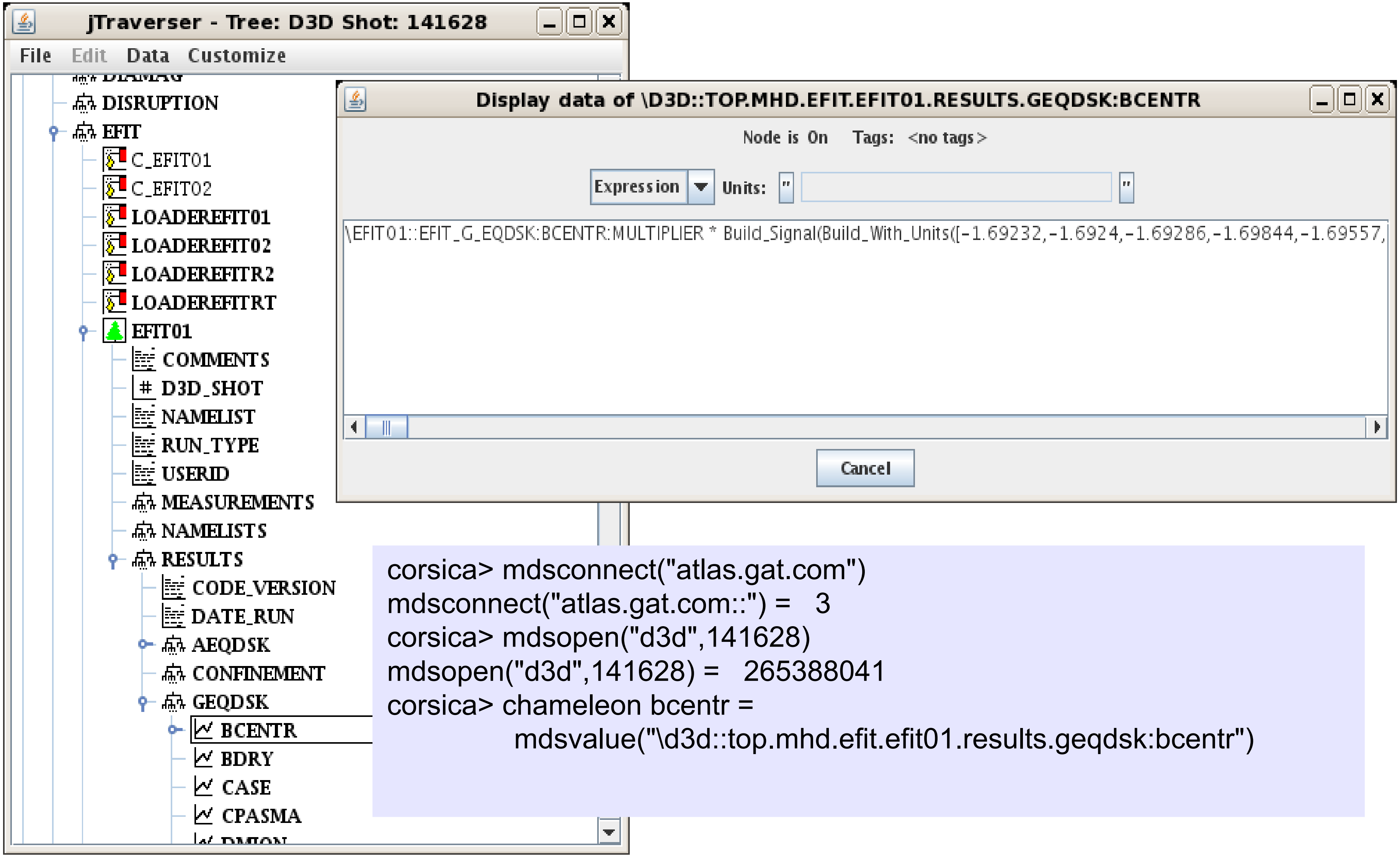
# MDS – MDSplus

- Software for data acquisition, storage, and management of scientific data.
  - Self-descriptive
  - Hierarchical
  - Simple programming interface
  - Thick and thin client/server models

- "Developed jointly by the Massachusetts Institute of Technology, the Fusion Research Group in Padua, Italy (Istituto Gas Ionizzati and Consorzio RFX), and the Los Alamos National Lab, MDSplus is the most widely used system for data management in the magnetic fusion energy program. It is currently installed at over 30 sites spread over 4 continents. "

- http://www.mdsplus.org

# MDSplus DIII-D Tree



Node is On  Tags: <no tags>

Expression  Units: "  "

```
\EFIT01::EFIT_G_EQDSK:BCENTR:MULTIPLIER * Build_Signal(Build_With_Units([-1.69232,-1.6924,-1.69286,-1.69844,-1.69557,
```

```
corsica> mdsconnect("atlas.gat.com")
mdsconnect("atlas.gat.com::") =   3
corsica> mdsopen("d3d",141628)
mdsopen("d3d",141628) =   265388041
corsica> chameleon bcentr =
              mdsvalue("\d3d::top.mhd.efit.efit01.results.geqdsk:bcentr")
```

# MDS – Mdsplus for Corsica

- Light client interface
  - Socket connection to mdsplus server
    - Mdsconnect, mdsdisconnect
  - Tree navigation
    - Mdsopen, mdssetdefault
  - Events
    - Mdswfevent, mdssetupevent
  - Data
    - Mdsvalue, mdsput

- Higher level Basis script routines
  - Special purpose wrappers around (multiple) mdsplus routines
  - DIII-D gadat
  - MDSplus EFIT trees support
  - MDSplus server interface to callable IDL

# MDS – Flow

- Open network connection to server
  - mdsconnect(<string server specification>)
  - mdsconnect(“atlas.gat.com”)  # for DIII-D
  - mdsconnect(“mdsplus”)          # for llnl clusters
- Open tree/shot
  - mdsopen(<string tree name>,<int shot number>)
    - Opened on server and some information cached
      - To read modified tree may require mdsclose
  - mdsopen(“d3d”,141628)
- Set working node within tree
  - mdssetdefault(<path to tree node>)
  - mdssetdefault(“\d3d::top.mhd.efit.efit01.results”)
- Access data
  - Var = mdsvalue(“\d3d::top.mhd.efit.efit01.results.geqdsk:bcentr”)
  - Var = mdsvalue(“geqdsk:bcenter”)
    - With above mdssetdefault
  - Var = mdsvalue(“ptdata($,$)”,”ip”,141628)   # ptdata tdi function

# MDS – Getting DIII-D with ptdata and gadat

- MDSPlus server tree data interface (TDI) routine ptdata
  - Server side access to DIII-D data access library ptdata
    - Chameleon var = mdsvalue(“ptdata($,$)”,”ip”,141628)
    - Chameleon tvar = mdsvalue(“dim_of(__ptdata_signal)”)
- Basis script gadat.bas
  - Read gadat.bas
  - Chameleon var,tvar
  - gadat(“tvar”,”var”,”ip”,141628)
    - Note that variables names are passed as strings
- Only available on DIII-D MDSPlus server atlas.gat.com
  - Connect with mdsconnect(“atlas.gat.com”)

# MDS – Loading EFIT Results from MDSPlus

- DIII-D experiment only - atlas.gat.com
  - Implemented in scripts/DIII-D/mdsd3.bas
    - Converted from eqdsk file reader scripts/DIII-D/d3.bas
    - Eqdsk variables used from mds package
  - EFIT tree read by mdsreadefit from scripts/Mdsplus/mdseqdsk.bas
    - Not DIII-D specific but assumes the tree structure DIII-D implemented
    - All variables added as variables in mds package
- Read script mdsd3.bas
  - Defines mdsd3 and reads supporting scripts
- mdsreadefit(<str efit tree>, <int shot>, <int time ms>)
- mdsd3(<str efit tree>,.......)
  - Arguments same as scripts/DIII-D/d3.bas
  - Pass tree “help” for usage

# MDS - mdsreadefit

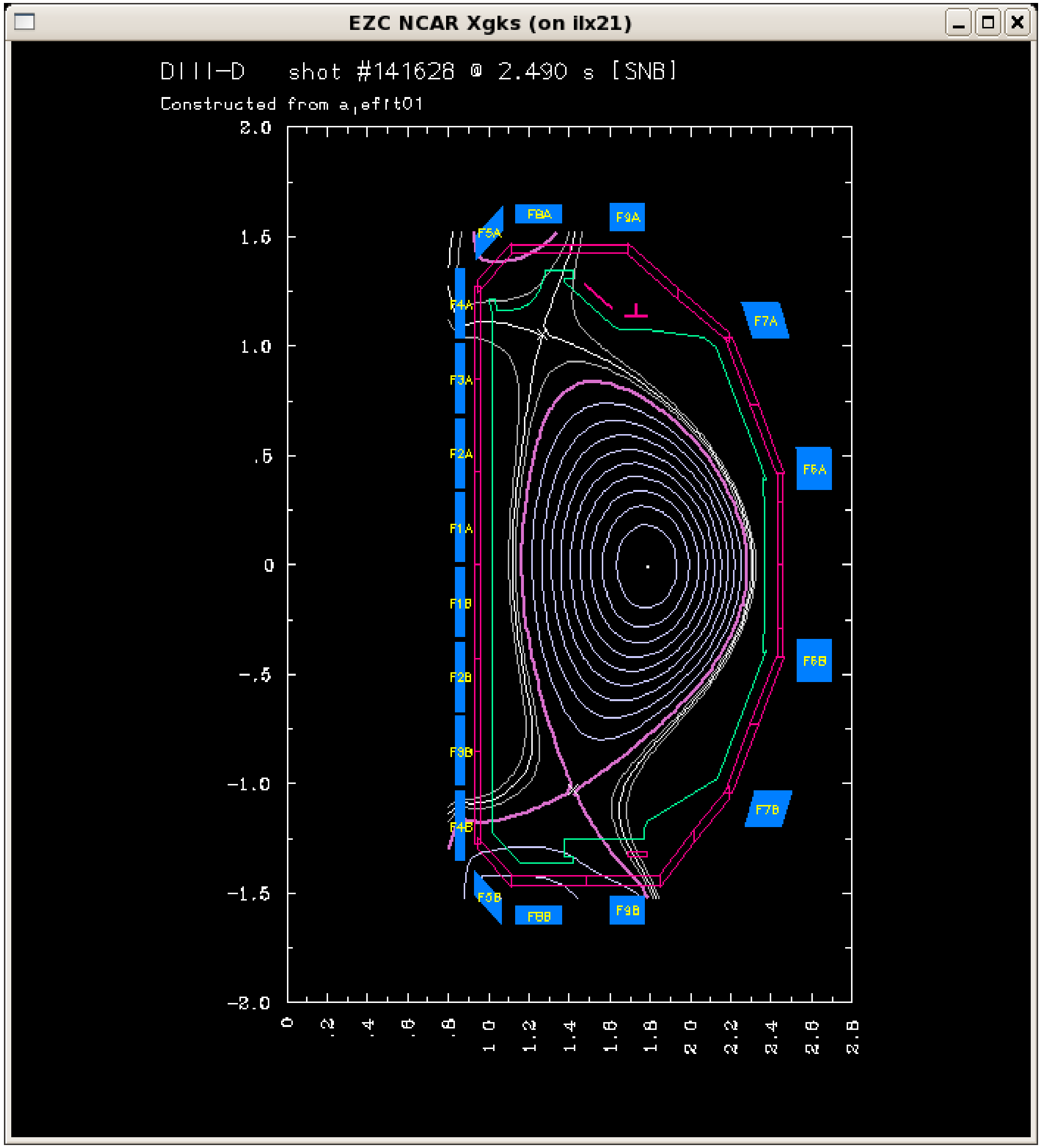
```
corsica> mdsconnect("atlas.gat.com")
mdsconnect("atlas.gat.com") =   3
corsica> read mdsd3.bas
corsica> mdsreadefit("efit01",141628,2490)
Read EFIT Aeqdsk tree
Read EFIT Geqdsk tree
Read EFIT Measurements tree
mdsreadefit("efit01",141628,2490) =   0
corsica> list mds.variables
MDSRestored:
shotnum  shottime shottree anids    apaths   anames   adtypes  x_       aaq2
ali      alpha    aminor   aout     area     atime    bcentr   betan    betap
betapd   betat    betatd   bpolav   bt0      bt0vac   cdflux   chigamt  chimse
chipre   chisq    cjor0    cjor95   condno   cprof    density  densr0   densv1
densv2   densv3   diamag   diamgc   diludom  diludomm dite     dminlx   dminux
dolubaf  dolubafm doutu    drsep    eout     error    fexpan   fexpvs   fit_type
gapbot   gapin    gapout   gaptop   in       indent   ipmeas   ipmhd    j0n
j1n      j95n     j99n     kappa    kappa0   li       li3      limloc   nebar_r0
nebar_v1 nebar_v2 nebar_v3 nindx    olefs    oleft    oring    otop     otops
.........
```

# MDS - mdsd3

```
corsica> mdsd3("efit01",0)
Restoring generic equilibrium: d3d65x65.sav
Getting Green's functions from greens65x65x140.pfb

PROBLEM NO. 1  ceq
 ihy  =   0 nceq =   2 msrf =  61 lsrf =  -1 thetac =  0.0000
 vo   =     cc(7)     cc(16)
 vo0  =    -5.74E-02 -6.48E-02
 vi   =     rbd(1)    rbd(2)
 x0   =     2.00E+02  2.13E+02
  2  1 axis(35, 32)= 1.753E+02,-9.648E-01 xpt(22,  7)= 1.391E+02,-1.235E+02 *
  1 cc(7)   =-5.7287d-02 (-5.7403d-02) <  0.011%>     rbd(1)  = 1.9994d+02
  2 cc(16)  =-6.4643d-02 (-6.4849d-02) <  0.019%>     rbd(2)  = 2.1343d+02

........

corsica> win; layout(0,0)
```

# MDS – Loading EFIT Results from MDSPlus

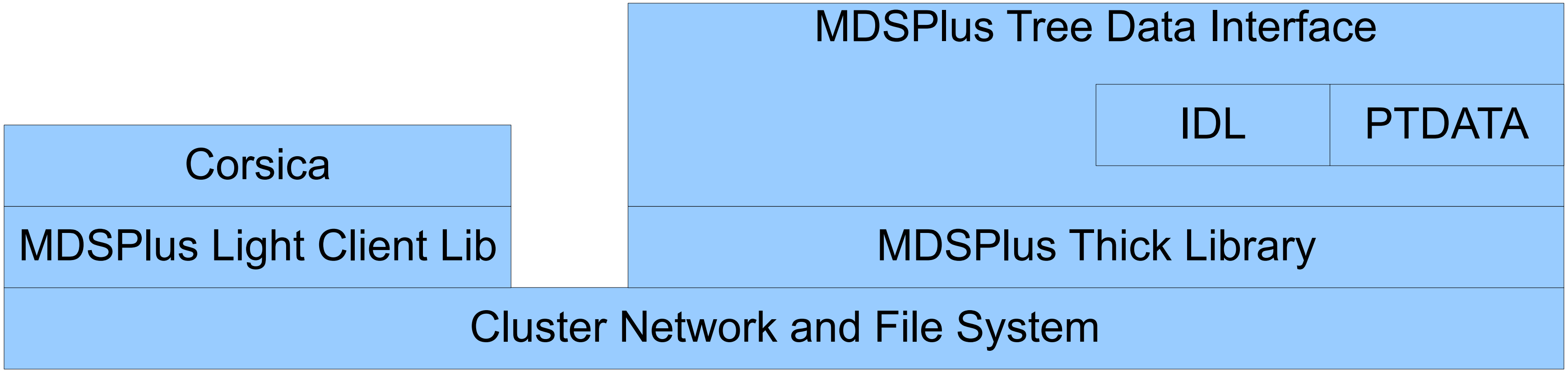
- Corsica has now written two save files 141628_2490.sav and 141628_2490_inv.sav
  - Same state as after running d3()
- Older shots either not loaded or incomplete
- MDSPlus intershot EFITs use more generic snapfiles
  - Must use geqdsk files and d3.bas for user generated EFIT



# MDS – Callable IDL

- IDL provides a programmatic interface which has been added to the LLNL FESP cluster mdsplus server
- Corsica Basis routines that wrap mds package calls
  - mdsidladdpath(<string>[:string]) – directory path for IDL to look for procedures
    - Set before first call to any other IDL routine
    - View path with mdsidlgetpath
  - mdsidl(<string>) - string executed by IDL parser on server
  - mdsidlexport(<string var name>) - returns IDL variable to corsica
  - mdsidlimportflt(<idlname>,<corsica var>) - puts Corsica data into IDL
    - Also mdsidlimportint and mdsidlimportdbl

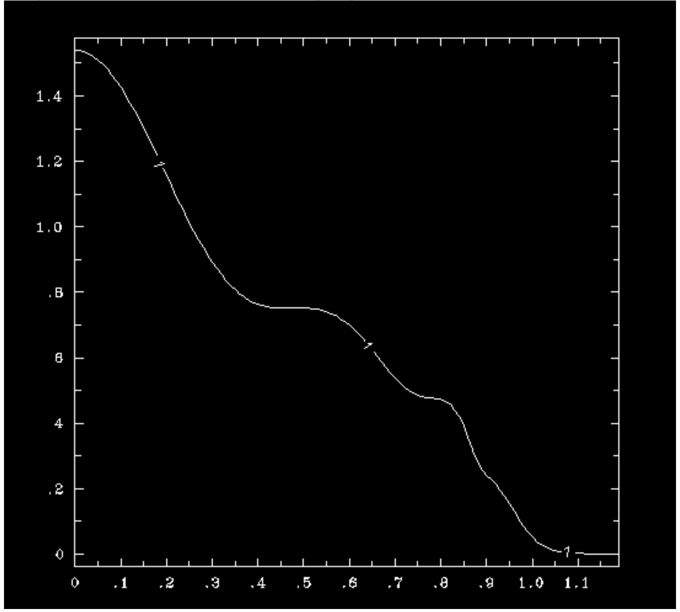
# MDS – Callable IDL



# MDS – Callable IDL Example

- **Use IDL zipfit routine to get DIII-D Te profile**

```
corsica> mdsconnect("mdsplus")
mdsconnect("mdsplus") =   3
corsica> read mdsplus.bas; read mdsidl.bas
Any calls to mdsidladdpath must be completed before Idl called
corsica> mdsidl("restore,'/afs/localcell/home/meyer8/zipfit.compile'")
corsica>  mdsidl("a =
fit_quick(141628,2490,/fit_edens,/fit_etemp,/dtfile,efit_type='EFIT01')")
corsica> mdsidl("restore,'dte141628.02490'")
corsica> chameleon rho_te = mdsidlexport("rho_te")
corsica> chameleon te = mdsidlexport("te")
corsica> win; plot te rho_te
```



# MDS - Summary

- Simple MDSPlus data routines work well for accessing DIII-D data
  - Both direct tree access and with tdi function ptdata
- Callable IDL has been used to get DIII-D profiles into Corsica
- MDSPlus events not covered by this talk
  - DIII-D events largely hidden from users
  - No current application so not completely tested
  - Not needed for control simulation application
- mdsput not covered by this talk
  - Seldom used except during development
  - Was used to record Corsica simulation time history for debug and post processing graphics
  - Not needed for control simulation application