Genie Module Interface

FAA, 20/3/96

This document will take you through the steps involved in creating, loading and executing a module from Genie.

1. Write a FORTRAN Subroutine to do your Task

The subroutine can take any parameters you desire. Our example will be based on:

```
C
C *** The function - just square the X array (from "myfunc.for")
C
      SUBROUTINE MYFUNC(X, Y, NPT)
      IMPLICIT NONE
      INTEGER NPT, I
      REAL X(NPT), Y(NPT)
      DO I=1,NPT
        Y(I) = X(I) * X(I)
      ENDDO
      RETURN
      END
C
```

2. Write a FORTRAN "wrapper" subroutine to interface between genie and your real subroutine

The wrapper must obey the following rules:

- It must be a SUBROUTINE taking just two parameters, called PARS_GET and PARS_PUT. These parameters should be declared EXTERNAL and not be accessed in any way except via "module_..." commands.
- It must allocate the variables needed by the user's SUBROUTINE and transfer data to/from them using "module_get_..." and "module_put_..." calls.
- As input/output devices are re-assigned by Genie, no READ(5,*) commands should be used and screen output should be handled via "module_print" and "module_information" commands.

Inside the module Genie variables are accessed by workspace field names. For example, in the program below the array X is loaded from the field labelled 'XVALS', where it has been placed by a GCL procedure. There is no restriction on how you label a variable, but often you will use the variable name itself. All "module_get_..." routines must specify PARS_GET as the first parameter, and similarly all "module_put_..." routines specify

PARS_PUT (these variables are, in fact, just pointers to the input/output parameter workspaces).

```
C
C *** the wrapper subroutine for MYFUNC
C *** input into arrtay X, output into array Y
C
      SUBROUTINE DO_MYFUNC(PARS_GET, PARS_PUT)
      IMPLICIT NONE
      INTEGER NPT_MAX, NPT
C *** We can handle NPT_MAX data points
      PARAMETER(NPT_MAX=1000)
      REAL X(NPT_MAX), Y(NPT_MAX)
      EXTERNAL PARS_GET, PARS_PUT
C *** print a "blue" message to the user
      CALL MODULE_INFORMATION('Inside module --- MYFUNC called!!!')
C *** Set NPT to the max size of the array
C *** On return, NPT will be the number of data points passed
      NPT = NPT_MAX
C *** The input was placed in a workspace field called 'XVALS' in GCL
      CALL MODULE_GET_REAL_ARRAY(PARS_GET, 'XVALS', X, NPT)
      IF (NPT .GT. NPT_MAX) THEN
        CALL MODULE_ERROR('myfunc module', 'Too many points passed', ' ')
        RETURN
      ENDIF
      CALL MYFUNC(X, Y, NPT)
C *** now return the result (Y) in a workspace field called 'YVALS'
      CALL MODULE_PUT_REAL_ARRAY(PARS_PUT, 'YVALS', Y, NPT)
      RETURN
      END
```

3. Make the Module Library

You need to compile the file containing the "function" and "wrapper" using the MODULE/COMPILE command --- this will produce a ".so" file to load into Genie.

```
>> MODULE/COMPILE "myfunc.for"
```

If the compilation fails for some reason you will see various error messages on the screen.

4. Load the module into genie

To load the module, use MODULE/LOAD with the name of the ".so" file created by MODULE/COMPILE. You can also supply a comment that will be displayed when you do a MODULE/LIST

>> MODULE/LOAD "myfunc.so" "My function module!"

You only need to re-load a module if you change the source code and execute another MODULE/COMPILE command.

5. Package function arguments into a workspace

You now need to create a workspace to hold the parameters of the SUBROUTINE you wish to call in the module. The fields in the workspace must have the same names as those you specified in the second argument of your "module_get_..." calls above. In our case, we only have one variable and we called it "XVALS", so assuming X is the array we wish to send:

```
>> PARS=FIELDS()          # create an empty workspace
>> PARS.XVALS = X         # add X to workspace as field "XVALS"
```

6. Execute the module

This is achieved by:

>> SOME_VARIABLE = MODULE:EXECUTE("my_func", PARS)

where "my_func" is the name of the FORTRAN wrapper routine you wrote, and PARS is the packaged arguments workspace. The values specified in "module_put_..." calls will appear in the returned workspace SOME_VARIABLE; in the above case, SOME_VARIABLE will contain one entry called YVALS, so we can type:

>> PRINTN SOME_VARIABLE.YVALS

and see out squared X values!

MODULE SUBROUTINES CALLABLE FROM FORTRAN

Commands to get Data from Genie

All commands to read data from Genie into a user program start with the "module_get_" prefix, and have their first two parameters in common. The first parameter, usually called PARS_GET, is an EXTERNAL entity passed by Genie. The second parameter, NAME, is a character variable which is the label assigned to the variable in GCL when the parameters workspace was created (XVALS in the above example). For arrays, a LENGTH variable must also be passed; on input this should be set to the maximum allowed number of points, and on output it will be set to the number actually read. If too many points are passed out from GCL, LENGTH will still indicate how many were passed, but the number actually

read will be the length of the array which was passed as input.; thus testing LENGTH on return provides a check for array overflow.

SUBROUTINE MODULE_GET_INT_ARRAY(EXTERNAL PARS_GET, CHARACTER*80 NAME, INTEGER VALUE(*), INTEGER LENGTH)

SUBROUTINE MODULE_GET_REAL_ARRAY(EXTERNAL PARS_GET, CHARACTER*80 NAME, REAL VALUE(*), INTEGER LENGTH)

SUBROUTINE MODULE_GET_STRING(EXTERNAL PARS_GET, CHARACTER*80 NAME, CHARACTER*512 VALUE)

SUBROUTINE MODULE_GET_REAL(EXTERNAL PARS_GET, CHARACTER*80 NAME, REAL VALUE)

SUBROUTINE MODULE_GET_INT(EXTERNAL PARS_GET, CHARACTER*80 NAME, INTEGER VALUE)

Commands to return Data to Genie

These commands all start with a "module_put_" prefix, and have the same form as the corresponding "module_get_" commands. The NAME parameter will be the field name in the result workspace returned to GCL, and the LENGTH parameter for the array routines will indicate how many data points to send.

SUBROUTINE MODULE_PUT_INT_ARRAY(EXTERNAL PARS_PUT, CHARACTER*80 NAME, INTEGER VALUE(*), INTEGER LENGTH)

SUBROUTINE MODULE_PUT_REAL_ARRAY(EXTERNAL PARS_PUT, CHARACTER*80 NAME, REAL VALUE(*), INTEGER LENGTH)

SUBROUTINE MODULE_PUT_STRING(EXTERNAL PARS_PUT, CHARACTER*80 NAME, CHARACTER*512 VALUE)

SUBROUTINE MODULE_PUT_REAL(EXTERNAL PARS_PUT, CHARACTER*80 NAME, REAL VALUE)

SUBROUTINE MODULE_PUT_INT(EXTERNAL PARS_PUT, CHARACTER*80 NAME, INTEGER VALUE)

Commands to print information

Print something to the screen

SUBROUTINE MODULE_PRINT(CHARACTER*(*) SOMETHING)

Print an INFORMATIONAL message (in blue) that will not appear if informational printing is turned off by TOGGLE/INFO

SUBROUTINE MODULE_INFORMATION(CHARACTER*(*) SOMETHING)

Return an error status to Genie, and print a message in RED

SUBROUTINE MODULE_ERROR(CHARACTER*(*) FUNCTION_NAME, CHARACTER*(*) ERROR_MESSAGE, CHARACTER*(*) POSSIBLE_SOLUTION)

A Real-Life Sample Module

The following "module" was created by Spencer Howells - the code is available in the "ex2" subdirectory of the examples area as "g2s.gcl" and "g2s.for"

First the GCL procedure that calls the module:

```
#
#  Procedures for G2S
#
PROCEDURE g2s
 PARAMETERS wg=workspace
 RESULT ws
 LOCAL g2s res
 ws=wg
 g2s=fields() ; g2s.lptin=wg.ntc
 g2s.Xin=wg.x ; g2s.Yin=wg.y ; g2s.Ein=wg.e
 g2s.qmax=inquire("g2s> qmax ")
 g2s.npt =inquire("g2s> number of points ")
 g2s.ic  =inquire("g2s> window function code ")
 g2s.rho =inquire("g2s> number density ")
 module/load "g2s.so" ; res=module:execute("g2s", g2s)
 ws.ntc=res.lptout
 ws.x=res.Xout ; ws.y=res.Yout ; ws.e=res.Eout
 ws.xlabel=res.xcaptout ; ws.ylabel=res.ycaptout
ENDPROCEDURE
```

Now the FORTRAN program that performs the actual calculation:

```
      SUBROUTINE G2S(g2s_get, g2s_put)
      EXTERNAL g2s_get, g2s_put
      INTEGER mn, lptin, lptout
      PARAMETER (mn=33000)
      REAL*4 Xin(mn),Yin(mn),Ein(mn),Xout(mn),Yout(mn),Eout(mn)
      real*4 xnew(mn),yw(mn)
      CHARACTER*40 xcaptout, ycaptout
      CHARACTER*10 char
      LOGICAL LMOD
      DANGLE=38.1*1.112/150.0/150.0
      PI=4.0*ATAN(1.0D0)

      call module_get_int(g2s_get, 'lptin', lptin)
      call module_get_real_array(g2s_get, 'Xin', Xin, lptin)
      call module_get_real_array(g2s_get, 'Yin', Yin, lptin)
      call module_get_real_array(g2s_get, 'Ein', Ein, lptin)
      call module_get_real(g2s_get, 'qmax', QMAX)
      call module_get_int(g2s_get, 'npt', npt)
```

```
          call module_get_int(g2s_get, 'ic', ichar)
          call module_get_real(g2s_get, 'rho', RHO)

          if(lptin.eq.0)then
            call module_error(" G2S",
          1 "ERROR  ** No input data", " ")
          endif

          if(npt.eq.0)then
            call module_error(" G2S>",
          1 "ERROR  ** Number of output points zero", " ")
            RETURN
          endif

          lptout=npt
         if(lptout.GT.mn)then
            call module_error(" G2S>",
          1  "#points reduced from 33000", " ")
            lptout=mn
           endif
          delq=QMAX/lptout

          if(rho.lt.1e-10)then
            call module_error(" G2S>",
          1 "ERROR  ** rho is zero", " ")
            RETURN
          endif

          if(ichar.eq.0)then
           LMOD=.false.
          else
           if(ichar.eq.1)then
            LMOD=.true.
           else
            LMOD=.false.
           endif
          endif
           if(LMOD)then
           A=PI/xin(lptin)
            do nn=1,lptin
           yw(nn)=SIN(xin(nn)*A)/xin(nn)/A
            end do
           else
            do nn=1,lptin
           yw(nn)=1.0
            end do
           endif
C
C  FORM VECTOR OF EQUALLY-SPACED R'S AT WHICH THE  FOURIER TRANSFORM
C  IS TO BE COMPUTED (RMAX IN ANGSTROMS), AND THE NUMBER OF R-POINTS.
C
           DO NR=1,lptout
            xout(NR)=delq*NR
          end do
C
C  THE NUMBER OF POINTS IN THE RANGE OF DATA TO BE TRANSFORMED.
C
```

```
          xd=(xin(2)-xin(1))/2.                              !half x-channel
          do n=1,lptin
c          xnew(n)=xin(n) +xd                      !offset - mid channel
           xnew(n)=xin(n)
           yw(n)=yw(n)*(yin(n)-1.)*xnew(n)
          end do
C
C  COMPUTE FOURIER TRANSFORM OF THE DATA
C
          delr=(xin(lptin)-xin(1))/(lptin-1)
        AFACT=delr*2.0/PI
        DO 35 NR=1,lptout
        FS=0.0
        RP=xout(NR)
         DO N=2,lptin
          SINUS1=SIN(xnew(N-1)*RP)
          SINUS=SIN(xnew(N)*RP)
            FS=FS+ (SINUS*yw(N)+
          1   SINUS1*yw(N-1))/2.0
           end do
        yout(NR)=FS*AFACT
35       CONTINUE
C


          pir=PI*PI*2.*RHO
          do n=1,lptout
           yout(n)=yout(n)*pir/xout(n) +1.
          end do

          xcaptout='    Q (Angstrom-1)     '
          ycaptout=' Structure Factor S of Q   '

          call module_put_int(g2s_put, 'lptout', LPTOUT)
          call module_put_real_array(g2s_put, 'Xout', Xout, lptout)
          call module_put_real_array(g2s_put, 'Yout', Yout, npt)
          call module_put_real_array(g2s_put, 'Eout', Eout, npt)
          call module_put_string(g2s_put, 'ycaptout', ycaptout)
          call module_put_string(g2s_put, 'xcaptout', xcaptout)
          call module_information(" G2S> output is in point mode")

          RETURN
          END
```