USING MSC/PATRAN FOR PRE- AND POSTPROCESSING FOR SPECIALIZED FEM CODES WHICH ARE NOT IN THE STANDARD MSC/PATRAN LIBRARY

DANIEL D. JENSEN

UNIVERSITY OF THE PACIFIC

DEPARTMENT OF MECHANICAL ENGINEERING

STOCKTON, CA 95211

ABSTRACT

MSC/PATRAN is commonly used in industry and academia as a pre- and postprocessor for commercially available FEM codes like MSC/NASTRAN, ANSYS, ABAQUS and others. However, a significant amount of analysis and research continues to be done with specialized FEM codes which do not have built in interfaces to MSC/PATRAN or any other widely available pre- or post processor. The present work provides a basic interface which allows models built in PATRAN access to the data necessary to build standard input decks for specialized FEM codes which are not supported by PATRAN. In addition, details are given for importing the results from analysis done with a specialized code back into PATRAN for visualization. An example is given which shows the ease of use of the interface. The interface presented provides an extremely expedient solution to the alternative of writing your own pre- and postprocessor.

INTRODUCTION

In many cases a significant percentage of the time spent on a FEM analysis is devoted to the pre- and post processing stages. Use of the pre- and postprocessor MSC/PATRAN is common for large commercial codes like MSC/NASTRAN, ANSYS, ABAQUS and others. In these cases a built in, full featured graphical user interface (GUI) is provided as part of the software. However, for specialized FEM codes which are not supported by PATRAN or any other widely available pre- and postprocessor, a method for pre- and postprocessing of the FEM data and results must be created. This is the case for numerous FEM codes used to do specialized research, to perform specific industrial analysis or for teaching finite element theory in the university. In order to handle the pre- and postprocessing duties in these cases, either a pre- and postprocessing package must be written from scratch or an interface must be created between the specialized code and a commercial pre- and postprocessor. Writing this type of software from scratch can be a time consuming, tedious process and often results in code which is neither robust nor full featured. If the decision is made instead to interface the FEM code with a commercial pre- and postprocessor, the decisions remaining are which commercial software to use and what complexity of interface to create. The present work provides a very basic interface which allows models built in PATRAN access to the data necessary to build standard input decks for the specialized FEM codes. The interface is basic in the sense that it does not contain a GUI interface and creates output information in ASCII format to be read by the specialized FEM code. The software which makes up this interface is built using the high level language PCL (PATRAN Command Language) which can be compiled directly from the PATRAN desktop. The interface is created to be an inexpensive and time efficient manner of providing access to the model building capabilities in MSC/PATRAN. Source code for the interface is provided so that it may be tailored for use with different specialized FEM codes. In addition, details are given for importing the results from analysis done with a specialized code back into PATRAN for visualization. It should be mentioned that MSC has a week long training class (Course

number is PAT 305) which gives instruction to those already familiar with use of the PCL language on creation of a more full featured, GUI interface between PATRAN and a specialized FEM code.

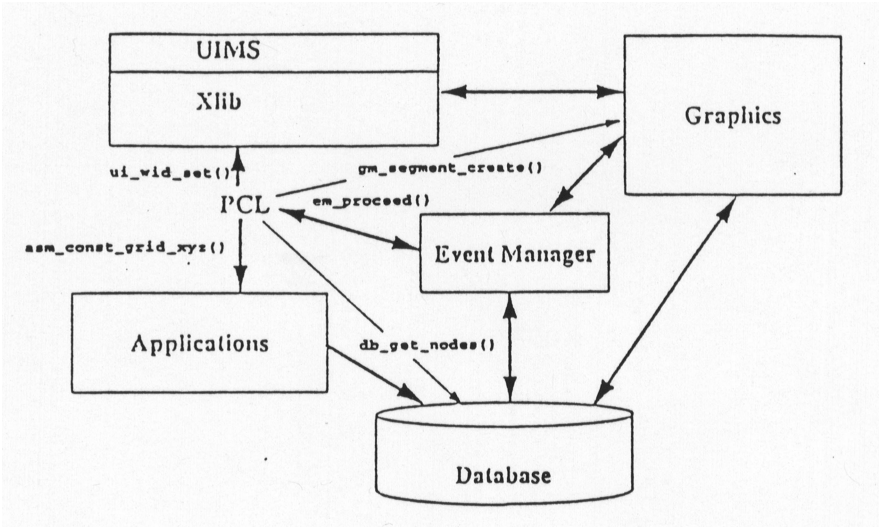
In order to provide context for customization of the of the interface, an overview of the structure of the PATRAN code as well as some information concerning the PCL language are given below. Specific instructions on use of the software directly from the PATRAN desktop are provided. In addition, an example is given which shows the flow and formats of the output as they are used with the author's personal research code. The method for reading results back in for visualization will also be demonstrated using this same example. Source code for the interface is included is Appendix I.

PCL AND THE STRUCTURE OF MSC/PATRAN

The overall architecture of PATRAN, as well as the role of the Patran Command Language (PCL), are shown below in Figure 1 [1]. The role of PCL in the work done by PATRAN could be thought of as providing macros for the types of tasks which need to be performed in interactions between the X library (XLib), the Events Manager, Applications and the Database. The present interface specifically uses PCL to interact with the Database. The goal of this interaction is to access the model data needed as input for a specialized FEM code. Detailed descriptions of the PCL commands needed to access this data are given in the on-line help for PATRAN as well as in hard copies of the manuals [1][5][6].

FIGURE 1

PATRAN ARCHITECTURE



USE OF THE INTERFACE "PAT_FEM"

The code, given in Appendix I, is designed to give the "Element Connectivity", "Nodal Coordinates", "Displacement Constraints" and Nodal Forces" as well as the normal mesh data such as the number of elements and number of nodes per element. The source code is well documented in order to make the code as easy as possible to manipulate for use with other specialized FEM codes. As PCL code can be compiled and run directly from the PATRAN desktop, use of the interface is not difficult. Steps for use are given below.

Steps for use of the interface PAT_FEM:

1) Either create a new database and build the model of the system to be analyzed or open an existing database for a model which was previously created. This step insures that you have an active database for PAT_FEM to access.

2) Go to the command window in PATRAN (lower window) and compile the PCL function by typing

!! input pat_fem.pcl

in the command line in the command window.

3) If previous runs have created previous output files, rename these files so that there are no files with the name pat_fem.out in the present working directory. Run the PCL code from the command line in the command window by typing the command

pat_fem()

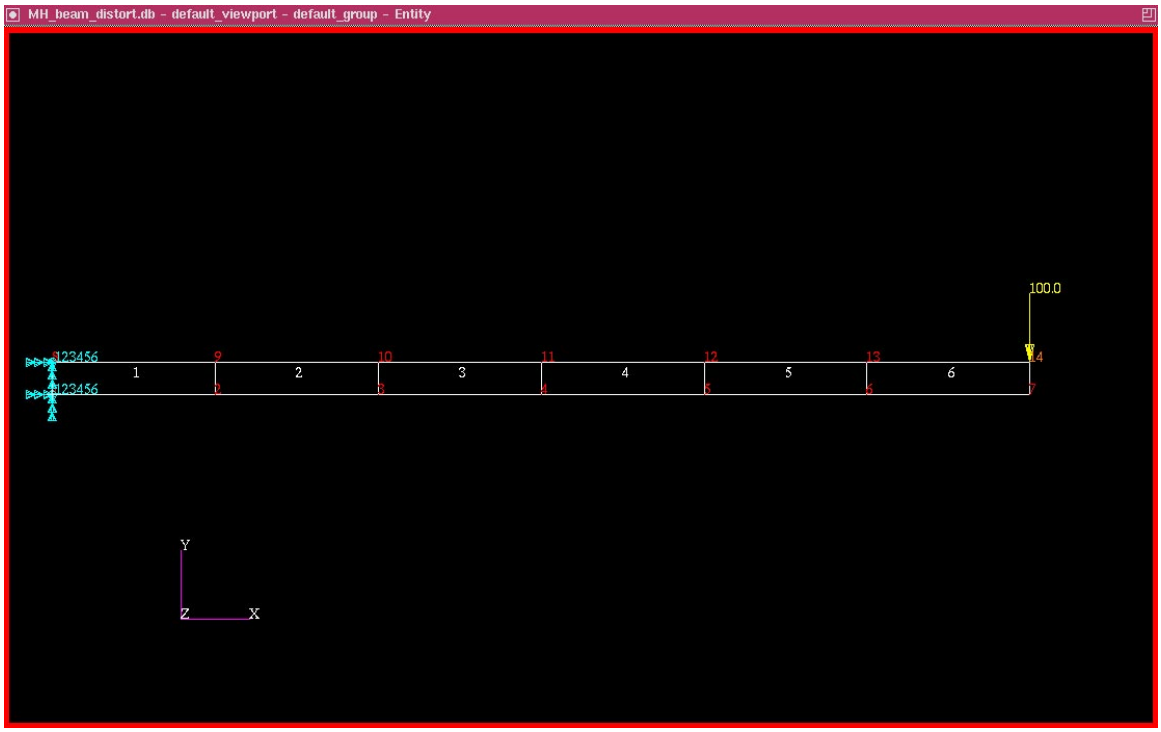
This will dump the data to be used as input for the external FEM code to a file called pat_fem.out.

A SIMPLE EXAMPLE

In order to exemplify the relationship between the model and the form of the output produced by PAT_FEM, a simple example from the test cases proposed by MacNeal and Harder [2] is chosen. The model has 6 elements each of which is a 4-node quad. The left side of the structure is fully constrained in all 6 dof and the right side is loaded in the -Y direction with a point force of 100.0. Relative size of the beam is chosen to be length=6.0, width=0.2 and thickness=0.1. Material properties are $E=1\times10^{7}$ and $\upsilon = 0.3$. The problem is shown below in Figure 2.

FIGURE 2

SAMPLE PROBLEM FOR TESTING PAT_FEM



The output created by running PAT_FEM as described above on the model shown in Figure 2 is given in Table 1. This is the file pat_fem.out.

TABLE 1

OUTPUT FILE PAT_FEM.OUT CREATED BY PAT_FEM.PCL

Element Connectivity
Format = Elem Number Connectivity
1 1 2 9 8
2 2 3 10 9
3 3 4 11 10
4 4 5 12 11
5 5 6 13 12
6 6 7 14 13

Nodal Coordinates
Format = Node Number X-coor Y-coor Z-coor
1 0. 0. 0.
2 0.9999997 0. 0.
3 1.9999996 0. 0.
4 2.9999998 0. 0.
5 4. 0. 0.
6 4.9999995 0. 0.
7 6. 0. 0.
8 0. 0.2 0.
9 0.9999997 0.2 0.
10 1.9999998 0.2 0.
11 2.9999998 0.2 0.
12 4. 0.2 0.
13 4.9999995 0.2 0.
14 6. 0.2 0.

Mesh Data
Format: number of elem nodes per elem
6 4

Displacement Constraints

Node	Ux	Uy	Uz	Rx	Ry	Rz
1	0.	0.	0.	--	--	--
8	0.	0.	0.	--	--	--
1	--	--	--	0.	0.	-0.
8	--	--	--	0.	0.	-0.

Forces on the Nodes

Node	Fx	Fy	Fz	Mx	My	Mz
14	0.	-100.	0.	--	--	--

POSTPROCESSING RESULTS FROM A SPECIALIZED FEM CODE

As a completely separate issue from the use of the interface PAT_FEM (which produces an ASCII output file with the model's data) the results of the analysis from the external specialized FEM code can be read back into PATRAN for visualization. For example, a specialized FEM analysis code may be used to find the displacements for the previously defined problem. These displacements (or stresses or strains etc.) must be in a specific format for import into PATRAN. The formats for the different types of results are found in [4] and [5]. For example, format for the displacements takes the form shown in Table 2.

TABLE 2
INPUT FORMAT FOR READING DISPLACEMENTS BACK INTO PATRAN

Line 1: title	data form = (80A1)
Line 2: nnodes, maxnod, defmax, ndmax, nwidth	data form = (2I9, E15,6, 2I9)
Line 3: Subtitle #1	data form = (80A1)
Line 4: Subtitle #2	data form = (80A1)
Line 5 to (nnodes + 4) : nodid, displ_x, displ_y, displ_z	data form = (3E13.7)

The variables above are defined as: nnodes = the number of nodes in the structure, maxnod = the highest node number in the structure, defmax = the maximum absolute displacement, ndmax = the node ID where the maximum displacement occurs, and nwidth = the number of columns after nodid for displacement information (3 if only displacements are read, 6 if rotations are included as well). The actual file for input back into PATRAN based on displacements from the previous beam structure is given below in Table 3.

TABLE 3

INPUT FILE FOR READING DISPLACEMENTS BACK INTO PATRAN

```
jensen_title_dispmh
     14      14       1.0E-01       7      3
subtitle1_dispmh
subtitle2_dispmh
     1     0.0E+00    -0.0E+00     0.0E+00
     2    -8.3E-02    -4.1E-01     0.0E+00
     3    -1.5E-01    -1.6E+00     0.0E+00
     4    -2.0E-01    -3.3E+00     0.0E+00
     5    -2.4E-01    -5.6E+00     0.0E+00
     6    -2.6E-01    -8.1E+00     0.0E+00
     7    -2.7E-01    -1.1E+01     0.0E+00
     8    -0.0E-00    -0.0E+00     0.0E+00
     9    -8.3E-02    -4.1E-01     0.0E+00
    10    -1.5E-01    -1.6E+00     0.0E+00
    11    -2.0E-01    -3.3E+00     0.0E+00
    12    -2.4E-01    -5.6E+00     0.0E+00
    13    -2.6E-01    -8.1E+00     0.0E+00
    14    -2.7E-01    -1.1E+01     0.0E+0
```

The process for reading a specialized FEM code output file back into PATRAN for visualization is as follows:

- Arrange the output data in the proper format [4][5]
- Select the IMPORT option under the FILE menu from the upper tool bar of the PATRAN desktop.
- Under FILE / IMPORT set Option = Results
- Select the FORMAT that corresponds to the type of data you are reading in (for displacement data this is the Patran 2. dis option)
- Select a template that corresponds to the format of the data being read in (for the displacement data above, the template is MSCNASTRAN_dis.res_tmpl)
- The Command window will confirm that your results have been read in.
- To visualize the imported quantities, follow the standard procedures for visualization of results using PATRAN.

When the displacements as given in Table 3 above are read back into PATRAN the displacements can be visualized as shown below in Figure 3.

FIGURE 3

VISUALIZATION OF DISPLACEMENT RESULTS READ BACK INTO PATRAN

CONCLUSIONS

An interface has been developed which allows the use of MSC/PATRAN as a pre- and postprocessor for specialized FEM codes which are not in the standard PATRAN library. The interface can be compiled and run directly from the PATRAN desktop. The PATRAN database is accessed by the interface and data necessary to build an input file for the specialized FEM code is written to an ASCII file. This interface would prove to

be an expedient solution for those using specialized FEM codes who do not wish to write their own pre- or postprocessor. With this in mind, source code is provided which may be modified for use with a variety of specialized FEM analysis codes. In addition, basic information for reading results from the specialized code back into PATRAN for visualization are given. A simple example which shows both the ability of the interface to access the database and download the FEM data as well as the ability to read the analysis results back into PATRAN for visualization has been included.

ACKNOWLEDGMENTS

The development of the interface has been partially funded through a grant from the Eberhardt Research Awards. Software and training have been provided for the University of the Pacific Engineering faculty through a grant from the MSC Corporation.

REFERENCES

[1] MSC Institute of Technology, "PAT 304 Customization Course Notes", The MacNeal - Schwendler Corporation, Costa Mesa, CA, pp. 3-3, 1994.

[2] R.H. MacNeal, R.L. Harder, "A Proposed Set of Problems to Test Finite Element Accuracy", Finite Elements in Analysis and Design, Vol. 11, pg. 3-20, 1985.

[3] O.C. Zienkiewicz, The Finite Element Method, 4th Ed, McGraw Hill, London, 1992.

[4] MSC Corporation, "Patran Plus User Manual", The MacNeal - Schwendler Corporation, Costa Mesa, CA, Sept., Pg.27-193., 1989.

[5] The MacNeal - Schwendler Corporation, "MSC/NASTRAN Interface Guide", MSC Los Angeles, CA, Appendix E.

[6] P3/PATRAN Users Manual, Release 1.3, Publication number 903000, X MSC Corporation, Costa Mesa, CA, December, 1993.

APPENDIX I -- SOURCE CODE FOR THE INTERFACE PAT_FEM

```
function pat_fem()


/* ************************************************
Purpose:
Obtain information regarding the model out of the PATRAN
data base for use in the external FEM code

Author: Daniel D. Jensen, University of the Pacific, Stockton, CA 95211
Date: Aug. 1995

Method for use:
* Create a PATRAN model
* Be sure that your model is up and running on p3 (so database exists)
* Go to the command window in p3 (lower window) and
        compile the PCL function by typing
        !! input pat_fem.pcl
        in the command line in the command window
* Run this PCL code from there by typing the command pat_fem()
        at the command line in the command window
* This will dump the data needed for the external FEM code to a
        file called pat_fem.out. Information included in this file
        will be the nodal connectivity array, the nodal locations, the
        boundary conditions and the loading conditions.

NOTE FOR HELP...
In order to get help interpreting or modifying the PCL code below, take the following steps:
        1. Click Help in the upper right corner of the P3 top Menu
        2. Select Document Directory on the subordinate menu
        3. Select PCL Reference Manual under Other Documents in the next form
        4. Click the box In the center close to the top labeled Index of Functions
        5. Click the box on the left labeled Part 9: PCL and Customization...
        This will provide detailed information on ALL of the functions called below

Local Variables:
        i, j  = counters for the loops
        elem_ids() = has the element ids for the elements
        connect() = connectivity for the elements
        num_elems = number of elements
        nodes_per_elem = nodes per ele
        chan_1 channel designation for the connectivity
        status = check for errors in function called
        elem_ids_str = string value of the element ids
        c_str1 - c_str7 = string values of various numbers
        filespec = string holder for name of the output file
        outstr = string holder for the connectivities
        topo_codes() = topology codes for the elements
        shape_codes()= element shape codes
        ref_coords() = ids of the reference coord frame
        analy_coords() = ids of the analysis coords
        glob_xyzs() = 3xnum_nodes w/ global rect. position
```

```
num_nodes = Number of nodes
node_ids() = node ids
node_ids_str() = string value of the nodal ids
gnc_strX - gnc_strZ = string values - global nodal coors
load_case_id = P3 id number for that load case
load_case_type = 1 for static, 2 for Time dependent, 3 for vibrations
num_loads = Number of loads for this load case
load_var_id = id of the loading type (for ex: force or moment) see the table
        on pg. 9-85 of the PCL and Customization documentation
entity_type = Indicates the type of entity a load or BC is applied to.
        123 --> node, 124 --> element, 153 --> element face, 154 --> element edge
entity_id = the number of the entity (node number
sub_entiy_id = id of the element face or edge (Zero if inappropriate)
node_position = Number of the node at which this value applies for variable loads only
region_type = applicable only for multiple application loads
num_values = Number of values for this load or BC. Note forces (displacements)
        and moments (rotations) are considered separately
load_type = P3 assigned number for this type of load (Ex: 6 for displacement/rotation BC
        7 foe force/moment loads)
application_type = indicates how the load/BC are applied (Ex: 1--> applied to nodes)
dynamic_flag = indicates if the load is constant
        (1--> static, 2--> frequency constant with time)
cid_flag = indicates if an alternate coordinate system is used.
        (0--> global coors used, >0--> alt. coords)
app_reg_order = Multiple application region option for load/BC
nodes_per_elem_array = contains the max nodes per elem for each elem type
load_ids = array (num_loads) of the internal p3 assigned ids of the loads
        (note this is NOT the same as the ID from the chart of pg. 9-85 of the PCL documents)
load_priorities = indicates the priority of loads for cases where >1
        load is applied on the same node
geo_fem = specifies whether load was originally appended to geom(=0) or fem (=1)
target_elem_dim = dimension of the elements loaded (1=lone, 2=surface, 3=solid)
null_vector = array (3) indicating if a vector component was left blank in the application
        ( -1 --> component is blank, 0--> component is specified)
evaluation_point = time or freq. at which a dynamic load is evaluated (=0 for static)
scale_factor = a scale factor which is already included in the load_value
load_value = the value of the load
load_case_name = name supplied by the user for the load case during model building
load_case_description = string containing description during model building
dynamic_case_name = name of the dynamic case from which a static case is derived
load_name = external name used to reference the load

**************************************************/

/* Declare the local variables */
integer nodes_per_elem, i, j, chan_1, status, num_elems, num_nodes
integer load_case_id, load_case_type, num_loads
integer load_var_id, entity_type, entity_id, sub_entity_id
integer node_position, region_type, num_values
integer load_type, application_type, dynamic_flag
integer cid_flag, acid_mod, app_reg_couple, app_reg_order, equiv_flag

integer elem_ids(virtual)
integer connect(virtual)
integer topo_codes(virtual)
```

```
integer shape_codes(virtual)
integer nodes_per_elem_array(virtual)
integer ref_coords(virtual)
integer analy_coords(virtual)
integer node_ids(virtual)
integer load_ids(virtual)
integer load_priorities(virtual)

integer geo_fem(2)
integer target_elem_dim(2)
integer null_vector(3)

real evaluation_point, scale_factor

real glob_xyzs(virtual)

real load_value(3)

string elem_ids_str[9]
string c_str1[9]
string c_str2[9]
string c_str3[9]
string c_str4[9]
string c_str5[9]
string c_str6[9]
string c_str7[9]
string node_ids_str[9]
string gnc_strx[9]
string gnc_stry[9]
string gnc_strz[9]
string filespec[80]
string outstr[100]
string load_case_name[80]
string load_case_description[80]
string dynamic_case_name[80]
string load_name[80]

/*****************************************
Find information needed to get connectivity
*****************************************   */

/* Find the number of elements in the database */
status = db_count_elems (num_elems)
dump num_elems

/* Allocate memory for the element manipulation */
sys_allocate_array(nodes_per_elem_array,1,num_elems)
sys_allocate_array(shape_codes,1,num_elems)
sys_allocate_array(topo_codes,1,num_elems)
sys_allocate_array(elem_ids,1,num_elems)

/* Get the element ids */
status = db_get_elem_ids(num_elems, elem_ids)
dump elem_ids
```

```
/* Get the elem topology */
status = db_get_elem_etop (num_elems, elem_ids, topo_codes)
dump topo_codes

/* Get the number of nodes per element */
status = db_get_elem_topology_data(num_elems,topo_codes, @
            shape_codes,nodes_per_elem_array)
nodes_per_elem = nodes_per_elem_array(1)
dump nodes_per_elem
dump nodes_per_elem_array
dump shape_codes

/* allocate memory for the connectivity */
sys_allocate_array(connect,1,nodes_per_elem*num_elems)

/* Get the node's connectivity for this mesh */
status = db_get_nodes_for_elems(num_elems,nodes_per_elem,elem_ids,connect)
dump connect


if (nodes_per_elem == 4) then

  /* ************************************************************
  Write the connectivities for a mesh of 4 node quads to the file
  of 4 node quads to a file named fem_conn.out
  ************************************************************ */

  /* Write the connectivity to a file */
  file_build_fname("","pat_fem","out","N",filespec)

  text_open(filespec,"NW",0,0,chan_1)
  text_write_string(chan_1,"Element Connectivity")
  text_write_string(chan_1,"Format = Elem Number   Connectivity")

  /* loop over the elements */
  for (i=1 to num_elems)
        elem_ids_str = str_from_integer(elem_ids(i))
        c_str1 = str_from_integer( connect( (i-1)*4+1 ) )
        c_str2 = str_from_integer( connect( (i-1)*4+2 ) )
        c_str3 = str_from_integer( connect( (i-1)*4+3 ) )
        c_str4 = str_from_integer( connect( (i-1)*4+4 ) )
        outstr = elem_ids_str// @
              "   "//c_str1//@
                        " "//c_str2//@
                        " "//c_str3//@
                        " "//c_str4
     text_write_string(chan_1,outstr)
     dump(outstr)
  end for


end if


/*******************************************
```

```
 Find information needed to get node locations
 ********************************************   */

/* get the number of nodes */
status = db_count_nodes(num_nodes)
dump num_nodes


/* Allocate memory for the nodal manipulation */
sys_allocate_array(node_ids,1,num_nodes)
sys_allocate_array(ref_coords,1,num_nodes)
sys_allocate_array(analy_coords,1,num_nodes)
sys_allocate_array(glob_xyzs,1,num_nodes,1,3)

/* get the node's ids */
status = db_get_node_ids(num_nodes, node_ids)
dump node_ids

/* get the nodal coords */
status = db_get_nodes (num_nodes, node_ids, ref_coords, @
                       analy_coords, glob_xyzs)
dump ref_coords
dump analy_coords
dump glob_xyzs

/* ************************************
 Write the node locations for this mesh
  ************************************  */
/* file_build_fname("","pat_fem","out","NO",filespec) */

/* text_open(filespec,"A",0,0,chan_1) */

text_write_string(chan_1,"  ")
text_write_string(chan_1,"Nodal Coordinates")
text_write_string(chan_1,"Format = Node Number X-coor  Y-coor  Z-coor")

/* loop over the nodes*/
for (i=1 to num_nodes)
      node_ids_str = str_from_integer(node_ids(i))
      gnc_strx = str_from_real( glob_xyzs( i,1 ) )
      gnc_stry = str_from_real( glob_xyzs( i,2 ) )
      gnc_strz = str_from_real( glob_xyzs( i,3 ) )
      outstr = node_ids_str// @
                  "   "//gnc_strx//@
                    " "//gnc_stry//@
                    " "//gnc_strz
      text_write_string(chan_1,outstr)
      dump(outstr)
end for

c_str1 = str_from_integer(num_elems)
c_str2 = str_from_integer(nodes_per_elem)
outstr = c_str1//" "//c_str2
text_write_string(chan_1,"     ")
text_write_string(chan_1,"Mesh Data")
```

```
text_write_string(chan_1,"Format: number of elem   nodes per elem")
text_write_string(chan_1,outstr)



/* ************************************
 Get the loads and bc from the database
****************************************** */

/* Find type active load case name */
status = db_get_active_load_case(load_case_name)

/* Count the number of loads for array size allocation */
status = db_count_lbc_by_load_case(load_case_name,num_loads)


/* Allocate the arrays */
sys_allocate_array(load_ids,1,num_loads)
sys_allocate_array(load_priorities,1,num_loads)

/* Get the load case id etc. for this load case */
status = db_get_load_case(load_case_name, load_case_id,load_case_type, @
          load_case_description, num_loads, load_ids, dynamic_case_name, @
          evaluation_point, load_priorities)

dump(load_case_name)
dump(load_case_id)
dump(load_case_type)
dump(load_case_description)
dump(num_loads)
dump(load_ids)
dump(dynamic_case_name)
dump(evaluation_point)
dump(load_priorities)


/*  Organize the load data */
status = loadsbcs_eval_all()

/* Loop over the number of types of loads/BCs for this load case */
/* Forces & moments, displacements & rotations, pressures etc. are each types */

for (i=1 to num_loads)

  status = db_get_lbc_new(load_ids(i), load_name, application_type, load_type, @
                target_elem_dim, dynamic_flag, cid_flag, scale_factor, geo_fem, @
                app_reg_couple, app_reg_order, equiv_flag, acid_mod)


  dump(load_name)
  dump load_type
  dump(application_type)
  dump(dynamic_flag)
```

```
/* Write the header for this lbc case */
 if (load_type == 6)  then
          text_write_string(chan_1,"  ")
          text_write_string(chan_1,"Displacement Constraints")
          text_write_string(chan_1,"Node    Ux          Uy          Uz"//@
          "           Rx          Ry          Rz")
 endif

 if (load_type == 7) then
          text_write_string(chan_1,"  ")
          text_write_string(chan_1,"Forces on the Nodes")
          text_write_string(chan_1,"Node    Fx          Fy          Fz"//@
          "           Mx          My          Mz")
 end if

 /* For use in controlling inner loop  */
 status = db_get_lbc_fem_count(load_ids(i), num_values)
 dump(load_ids(i))
 dump(num_values)

 status = db_get_all_fem_sv_by_id(load_ids(i))


 /* Loop over the number of discrete applications of this load */
 /* forces (or displacements) and moments (or rotations) are separate applications */
 for (j=1 to num_values)
          status = db_get_next_fem_sv_by_id(load_var_id, entity_type,@
          entity_id, sub_entity_id, @
          load_value, null_vector, scale_factor, node_position, region_type)

          c_str1 = "-- "
          c_str2 = "-- "
          c_str3 = "-- "
          c_str4 = "-- "
          c_str5 = "-- "
          c_str6 = "-- "
          c_str7 = "-- "
          if (load_type == 6) then
            if (entity_type != 123) then
               text_write_string(chan_1," LOADS MUST BE APPLIED AT THE NODES !!! ")
            endif
            c_str1 = str_from_integer(entity_id)
            if (load_var_id == 1) then
              if (null_vector(1) == 0) then
                c_str2 = str_from_real(load_value(1))
              endif
              if (null_vector(2) == 0) then
                c_str3 = str_from_real(load_value(2))
              endif
              if (null_vector(3) == 0) then
                c_str4 = str_from_real(load_value(3))
              endif
            endif
            if (load_var_id == 2) then
              if (null_vector(1) == 0) then
```

```
      c_str5 = str_from_real(load_value(1))
    endif
    if (null_vector(2) == 0) then
      c_str6 = str_from_real(load_value(2))
    endif
    if (null_vector(3) == 0) then
      c_str7 = str_from_real(load_value(3))
    endif
  endif
endif

if (load_type == 7) then
  if (entity_type != 123) then
    text_write_string(chan_1," LOADS MUST BE APPLIED AT THE NODES !!! ")
  endif
  c_str1 = str_from_integer(entity_id)
  if (load_var_id == 1) then
    if (null_vector(1) == 0) then
      c_str2 = str_from_real(load_value(1))
    endif
    if (null_vector(2) == 0) then
      c_str3 = str_from_real(load_value(2))
    endif
    if (null_vector(3) == 0) then
      c_str4 = str_from_real(load_value(3))
    endif
  endif
  if (load_var_id == 2) then
if (null_vector(1) == 0) then
      c_str5 = str_from_real(load_value(1))
endif
    if (null_vector(2) == 0) then
      c_str6 = str_from_real(load_value(2))
    endif
    if (null_vector(3) == 0) then
      c_str7 = str_from_real(load_value(3))
    endif
  endif
endif
outstr =  " "//c_str1//@
          "       "//c_str2//@
          "       "//c_str3//@
          "       "//c_str4//@
          "       "//c_str5//@
          "       "//c_str6//@
          "       "//c_str7

text_write_string(chan_1,outstr)

dump(outstr)

dump(load_var_id)
dump(entity_type)
dump(entity_id)
dump(sub_entity_id)
```

```
            dump(load_value)
            dump(null_vector)
            dump(scale_factor)
            dump(node_position)
            dump(region_type)


  end for
end for


text_close(chan_1,"")

end function
```