Programmer's Manual for ANSYS

ANSYS Release 11.0

002328
January 2007
Copyright and Trademark Information

© 2007 SAS IP, Inc. All rights reserved. Unauthorized use, distribution or duplication is prohibited.

ANSYS, ANSYS Workbench, AUTODYN, CFX, FLUENT and any and all ANSYS, Inc. brand, product, service and feature names, logos and slogans are registered trademarks or trademarks of ANSYS, Inc. or its subsidiaries located in the United States or other countries. ICEM CFD is a trademark used by ANSYS, Inc. under license. All other brand, product, service and feature names or trademarks are the property of their respective owners.

Disclaimer Notice

THIS ANSYS SOFTWARE PRODUCT AND PROGRAM DOCUMENTATION INCLUDE TRADE SECRETS AND ARE CONFIDENTIAL AND PROPRIETARY PRODUCTS OF ANSYS, INC., ITS SUBSIDIARIES, OR LICENSORS. The software products and documentation are furnished by ANSYS, Inc., its subsidiaries, or affiliates under a software license agreement that contains provisions concerning non-disclosure, copying, length and nature of use, compliance with exporting laws, warranties, disclaimers, limitations of liability, and remedies, and other provisions. The software products and documentation may be used, disclosed, transferred, or copied only in accordance with the terms and conditions of that software license agreement.

ANSYS, Inc. and ANSYS Europe, Ltd. are UL registered ISO 9001:2000 Companies.

U.S. Government Rights

For U.S. Government users, except as specifically granted by the ANSYS, Inc. software license agreement, the use, duplication, or disclosure by the United States Government is subject to restrictions stated in the ANSYS, Inc. software license agreement and FAR 12.212 (for non DOD licenses).

Third-Party Software

See the online documentation in the product help files for the complete Legal Notice for ANSYS proprietary software and third-party software. The ANSYS third-party software information is also available via download from the Customer Portal on the ANSYS web page. If you are unable to access the third-party legal notices, please contact ANSYS, Inc.

Published in the U.S.A.
# Table of Contents

Preface ................................................................. xvi

## I. Guide to Interfacing with ANSYS

### 1. Format of Binary Data Files .......................................... 1

#### 1.1. What Are ANSYS Binary Files? ..................................... 3

##### 1.1.1. Conventions Used to Describe Binary Files ....................... 3

##### 1.1.2. The Standard Header for ANSYS Binary Files .......... 3

#### 1.2. Format of the Results File ......................................... 4

##### 1.2.1. Nomenclature ..................................................... 4

##### 1.2.2. Standard ANSYS File Header ..................................... 5

##### 1.2.3. Results File Format ............................................... 5

#### 1.3. Description of the Reduced Displacement File .................... 21

##### 1.3.1. Standard ANSYS File Header ..................................... 21

##### 1.3.2. RDSP File Format ................................................. 21

#### 1.4. Description of the Reduced Complex Displacement File ........ 25

##### 1.4.1. Standard ANSYS File Header ..................................... 25

##### 1.4.2. RFRQ File Format .................................................. 25

#### 1.5. Description of the Modal Results File .............................. 29

##### 1.5.1. Standard ANSYS File Header ..................................... 29

##### 1.5.2. MODE File Format .................................................. 29

#### 1.6. Description of the Element Matrices File ........................... 32

##### 1.6.1. Standard ANSYS File Header ..................................... 32

##### 1.6.2. EMAT File Format ................................................... 32

#### 1.7. Description of the Substructure Matrices File ..................... 36

##### 1.7.1. Standard ANSYS File Header ..................................... 37

##### 1.7.2. SUB File Format ................................................... 37

#### 1.8. Description of the Component Mode Synthesis Matrices (CMS) File ...... 41

##### 1.8.1. Standard ANSYS File Header ..................................... 41

##### 1.8.2. CMS File Format ................................................... 41

#### 1.9. Description of the Triangularized Stiffness File .................. 43

##### 1.9.1. Standard ANSYS File Header ..................................... 43

##### 1.9.2. TRI File Format ................................................... 43

#### 1.10. Description of the Full Stiffness-Mass File ....................... 46

##### 1.10.1. Standard ANSYS File Header ..................................... 46

##### 1.10.2. FULL File Format .................................................. 46

### 2. Accessing Binary Data Files ........................................... 53

#### 2.1. Accessing ANSYS Binary Files ..................................... 53

##### 2.1.1. Access Routines to Results and Substructure Files ............ 53

##### 2.1.2. Characteristics of ANSYS Binary Files .......................... 54

##### 2.1.3. Viewing Binary File Contents ..................................... 55

##### 2.1.4. Abbreviations ...................................................... 55

##### 2.1.5. binini (Initializing Buffered Binary I/O Systems) ............. 55

##### 2.1.6. Function sysiqr (Retrieving the Status of a File) ............. 55

##### 2.1.7. Function biniqr8 (Retrieving System-Dependent Parameters) .. 56

##### 2.1.8. Function binset (Opening a Blocked Binary File or Initializing Paging Space) ... 57

##### 2.1.9. Subroutine bintfo (Defining Data for a Standard ANSYS File Header) .... 57

##### 2.1.10. Subroutine bintkd (Writing the Standard ANSYS File Header) .... 58

##### 2.1.11. Subroutine binnr8 (Reading Data from a Buffered File) ....... 59

##### 2.1.12. Subroutine binwrt8 (Writing Data to a Buffered File) ........ 59

##### 2.1.13. Subroutine inxc4 (Decoding an Integer String into a Character String) ... 60

##### 2.1.14. Subroutine inexc4 (Coding a Character String into an Integer String) .... 60
### 2. Demonstration Routines

2.1. Program bintst (Demonstrates Dumping a Binary File and Copying It for Comparison Purposes)

2.2. Program wrtsub (Demonstrates Writing an ANSYS Substructure File)

2.3. Program rdsubs (Demonstrates Reading a Substructure File)

2.4. Program rdfull (Demonstrates Reading and Reformatting the .FULL File)

2.5. Program ResRdDemo (Demonstrates Reading a Results File)

2.6. Program ResWrDemo (Demonstrates Writing a Results File)

2.7. Program ResRdDemo (Demonstrates Reading a Results File)

2.8. Program ResWrDemo (Demonstrates Writing a Results File)

2.9. Program ResRdDemo (Demonstrates Reading a Results File)

2.10. Program ResWrDemo (Demonstrates Writing a Results File)

### 3. Results File Access Routines

3.1. Overview of the Routines

3.2. ResRdBegin (Opening the File and Retrieving Global Information)

3.3. ResRdGeomBegin (Retrieving Global Geometry Information)

3.4. ResRdType (Retrieving Element Types)

3.5. ResRdReal (Retrieving Real Constants)

3.6. ResRdCsyst (Retrieving Coordinate Systems)

3.7. ResRdNode (Retrieving Nodal Coordinates)

3.8. ResRdElem (Retrieving Elements)

3.9. ResRdSolBegin (Retrieving Result Set Location)

3.10. ResRdDisp (Retrieving Nodal Solution)

3.11. ResRdRfor (Retrieving Reaction Solution)


### 3. Using CDREAD and CDWRITE

3.1. Using the CDREAD Command

3.2. Using the CDWRITE Command

3.3. Coded Database File Commands

3.4. CE Command

3.5. CP Command

3.6. CMBLOCK Command

3.7. EBLOCK Command

3.8. EDCADAPT Command

3.9. EDCURVE Command

3.10. EDCGEN Command

3.11. EDDCURVE Command

3.12. EDDRELAX Command

3.13. EDLCS Command

3.14. EDLOAD Command

3.15. EDPREAD Command

3.16. EDWELD Command

3.17. EN Command

3.18. LOCAL Command

3.19. M Command

3.20. MPDATA Command

3.21. MPTTEMP Command

3.22. N Command
II. Guide to ANSYS User Programmable Features.................................................. .............................................. 99

5. Using User Programmable Features (UPFs) ................................................... 101
  5.1. What Are UPFs? ................................................... 101
  5.2. What You Should Know Before Using UPFs ................................................. 102
  5.3. Planning Your UPFs ............................................................................. 103
  5.4. Studying the ANSYS User Routines .................................................. 103
  5.5. Programming in Languages Other than FORTRAN .................................. 103
  5.6. Developing UPFs: a Suggested Strategy .............................................. 103
  5.7. Include Decks ................................................................................. 104
  5.8. Linking User Routines ........................................................................ 105
  5.9. Compiling and Linking UPFs on UNIX Systems ..................................... 105
  5.10. Compiling and Linking UPFs on Windows Systems ............................ 106
  5.11. Activating UPFs ............................................................................ 107
  5.12. Running Your Custom Executable ................................................... 108
  5.13. Verifying Your Routines .................................................................. 108
  5.14. Debugging Commands .................................................................. 109
      5.14.1. Tracking the Path of Program Logic ........................................... 109
      5.14.2. Debugging Elements and Solutions ........................................... 109
      5.14.2.1. Solution Debug Format ...................................................... 109
      5.14.2.2. Element Debug Format ..................................................... 110
      5.14.2.3. General Debug Format ..................................................... 111
  5.15. Other Useful Commands ............................................................... 111
  5.16. Generating Output ........................................................................ 112
  5.17. Reading Large Data Files More Rapidly ............................................. 112

6. UPF Routines and Functions .......................................................................... 113
  6.1. Creating a New Element .................................................................. 113
      6.1.1. Creating a New Element via the User-Defined Element API .......... 114
      6.1.1.1. Subroutine UserElem (Writing Your Own Elements) ............ 115
      6.1.1.2. Subroutine ElemGetMat (Calling the ANSYS Standard Structural Material Library) ... 115
      6.1.2. Creating a New Element by Directly Accessing the ANSYS Database .... 126
      6.1.2.1. Input and Output Abbreviations ............................................ 126
      6.1.2.2. User Routines .................................................................. 127
      6.1.2.3. Subroutine uec100 (Defining Characteristics of the usr100 Routine) ................ 127
      6.1.2.3.1. Subroutines uec101 through uec105 ................................ 128

4. ANSYS Graphics File Format ........................... 89
  4.1. Modifying ANSYS Graphics Files .................................................. 89
  4.2. Pixmap Format for Graphic Display Files ........................................... 89
  4.3. Neutral Graphics File Format .................................................. 90
      4.3.1. Characters the Graphics File Uses ........................................... 90
      4.3.2. Graphics File Directives .................................................. 91
          4.3.2.1. Parameter Types for Graphics File Directives .................. 91
          4.3.2.2. Directive Descriptions ................................................. 92
          4.3.2.3. Color Specification .................................................. 93
      4.3.3. Decoding a Graphics File: an Example .................................... 94
          4.3.3.1. The Example Command Stream ................................ 95
          4.3.3.2. Example Graphics File Contents ................................ 95

3.3.19. NBLOCK Command .................................................. 84
3.3.20. R Command .................................................. 85
3.3.21. RLBLOCK Command .................................................. 85
3.3.22. SECBLOCK Command .................................................. 86
3.3.23. SFBEAM Command .................................................. 87
3.3.24. SFE Command .................................................. 87
6.2. Supporting Subroutines for Element Creation ................................................................. 131
   6.2.1. Subroutine nminfo (Returning Element Reference Names) ........................................... 131
   6.2.2. Subroutine svgidx (Fetching the Index for Saved Variables) ........................................ 131
   6.2.3. Subroutine svrput (Writing an Element’s Saved Variable Set) ...................................... 132
   6.2.4. Subroutine svrput (Writing an Element’s Saved Variable Data for an Element) ............... 132
   6.2.5. Subroutine svpidx (Writing the Saved Variable Element Index to a File) ........................... 133
   6.2.6. Subroutine meuse (Determining Which Element Matrices Can Be Reused) ...................... 133
   6.2.7. Subroutine subrd (Reading Element Load Data for a Substructure Generation Run) .......... 134
   6.2.8. Subroutine subwr (Writing an Element Load Vector to a File for a Substructure Generation Run) .................................................................................................................. 135
   6.2.9. Subroutine rvrget (Fetching Real Constants for an Element) ......................................... 135
   6.2.10. Subroutine propev (Evaluating a Group of Material Properties) ..................................... 135
   6.2.11. Subroutine prope1 (Evaluating One Material Property) ............................................... 136
   6.2.12. Subroutine psteve1 (Evaluating Material Properties for 1-D Elements) ............................ 137
   6.2.13. Subroutine tbuser (Retrieving User Table Data) ......................................................... 137
   6.2.14. Subroutine plast1 (Updating an Element’s Plastic History) .......................................... 137
   6.2.15. Subroutine plast3 (Updating an Element’s Plastic History, 4 or 6 components) ............... 138
   6.2.16. Subroutine creep1 (Updating an Element’s Creep History) .......................................... 139
   6.2.17. Subroutine creep3 (Updating an Element’s Creep History, 3-D Elements) ....................... 139
   6.2.18. Subroutine swell1 (Updating an Element’s Swelling History) ....................................... 140
   6.2.19. Subroutine swell3 (Updating an Element’s Swelling History, 3-D Elements) .................... 140
   6.2.20. Function eLenPsvrBuf (Determining additional ESAV Record for Plasticity) ................... 141
   6.2.21. Function nlgert (Retrieving Material Nonlinear Property Information) ......................... 141
   6.2.22. Subroutine usere (Storing User-Supplied Element Output) .......................................... 142
   6.2.23. Subroutine updrot (Updating the Rotation Pseudovector) .......................................... 142
   6.2.24. Subroutine updwdrl (Writing Element Nonsummable Miscellaneous Data to the Results File) .................................................................................................................. 143
   6.2.25. Subroutine trrot (Computing the Rotation Vector) ....................................................... 143
   6.2.26. Subroutine rotr (Computing the Transformation Matrix) ............................................ 143
   6.2.27. Subroutine xyzup3 (Updating an Element’s 3-D Nodal Coordinates) ............................. 144
   6.2.28. Subroutine updrot (Updating the Rotation Pseudovector) .......................................... 144
   6.2.29. Subroutine tmpget (Defining Current Temperature Loads) ........................................... 144
   6.2.30. Subroutine prsget (Defining Current Pressure Loads) .................................................. 145
   6.2.31. Subroutine cnvget (Defining Current Convection Loads) ............................................. 145
   6.2.32. Subroutine hgnget (Defining Current Heat Generation Loads) ....................................... 146
   6.2.33. Subroutine prinst (Computing principal stress and stress intensity) .............................. 146
6.3. Routines for Modifying and Monitoring Existing Elements ................................................. 146
   6.3.1. Subroutine userfd (Computing the Complex Load Vector for Frequency Domain Logic) .... 147
   6.3.2. Subroutine userou (Storing User-Supplied Element Output) .......................................... 148
   6.3.3. Subroutine useran (Modifying Orientation of Material Properties) ................................. 149
   6.3.4. Subroutine usanly (Modifying Orientation of Material Properties and Stresses of Layers) .... 149
   6.3.5. Subroutine userrc (Performing User Operations on COMBIN7 and COMBIN37 Parameters) .................................................................................................................. 150
6.3.6. Function userpe (Calculating Rotation Caused by Internal Pressure) ........................................ 150
6.3.7. Subroutine UE IMatx (Accessing Element Matrices and Load Vectors) ........................................ 151
6.3.8. Subroutine UTHICK (Getting User-defined Initial Thickness) .................................................. 151
6.3.9. Subroutine USTRESS (Getting User-defined Initial Stress) ..................................................... 152
6.3.10. Subroutine UsrFictive (Providing User-defined Fictive Temperature Relationship) ................. 152
6.3.11. Subroutine UsrViscEl (Performs Viskoelastic Computation) .................................................. 153
6.3.12. Subroutine ussurf116 (Modifying SURF151 and SURF152 Film Coefficients and Bulk
Temperatures) .................................................................................................................................. 154
6.3.13. Subroutine User116Cond (Computes the conductance coefficient for FLUID116) ...................... 155
6.3.14. Subroutine User116Hf (Computes the film coefficient for FLUID116) ........................................ 155
6.3.15. Subroutine Us_Surf_Str (Captures surface stresses) ................................................................. 156
6.3.16. Subroutine usflex (Computes the flexibility factor for PIPE16, PIPE17, PIPE18, and PIPE60) ........................................................................................................................................ 156
6.3.17. Subroutine UsrShift (Calculates pseudotime time increment) .................................................. 157
6.4. Subroutines for Customizing Material Behavior ............................................................................ 157
6.4.1. Subroutine usermat (Writing Your Own Material Models) .......................................................... 157
6.4.2. Subroutine userpl (Writing Your Own Plasticity Laws) ............................................................. 160
6.4.3. Subroutines usercreep and usercr (Defining Viskoelastic/Creep Material Behavior) ................. 161
  6.4.3.1. Creep Subroutine usercreep ................................................................................................... 162
  6.4.3.2. Creep Subroutine usercr ...................................................................................................... 163
6.4.4. Subroutine usersw (Writing Your Own Swelling Laws) ............................................................. 164
6.4.5. Subroutine UserHyper (Writing Your Own Hyperelasticity Laws) ............................................ 165
6.4.6. Subroutine uservp (Updating Nonlinear Strain History for Materials) ....................................... 166
6.4.7. Subroutine userck (Checking User-Defined Material Data) ....................................................... 167
6.4.8. Subroutine usermc (Controlling Hydrothermal Growth) ......................................................... 167
6.4.9. Subroutine usrfc6 (Defining Custom Failure Criteria) ............................................................... 168
6.4.10. Subroutines usrfc1 through usrfc5 .......................................................................................... 168
6.4.11. Subroutine UserVisLaw (Defining Viskoelastic Laws) ............................................................. 168
6.4.12. Supporting Function egen ..................................................................................................... 169
6.5. Subroutines for Customizing Loads .............................................................................................. 170
6.5.1. Subroutine usrefl (Changing Scalar Fields to User-Defined Values) ........................................ 170
6.5.2. Subroutine userpr (Changing Element Pressure Information) ................................................ 171
6.5.3. Subroutine usercv (Changing Element Face Convection Surface Information) ...................... 171
6.5.4. Subroutine userfx (Changing Element Face Heat Flux Surface Information) ......................... 172
6.5.5. Subroutine userc (Changing Element Face Charge Density Surface Information) ................ 173
6.6. Subroutine as a Subroutine .......................................................................................................... 174
6.7. Defining Your Own Commands ................................................................................................... 174
6.7.1. Function user01 ......................................................................................................................... 175
6.7.2. Function user02 (Demonstrates Offsetting Selected Nodes) .................................................... 176
6.7.3. Function user03 (Demonstrates Using ANSYS Memory) ......................................................... 177
6.7.4. Function user04 ....................................................................................................................... 179
6.7.5. Functions user05 through user10 ............................................................................................. 181
6.8. Supporting Subroutines .............................................................................................................. 181
6.8.1. Function GetRForce (Getting Nodal Reaction Force Values) ................................................. 181
6.8.2. Function GetStackDisp (Getting Current Displacement Values) ............................................ 181
6.8.3. Subroutine ElResultStrt (Getting Load Data from Analysis Results) ....................................... 182
6.8.4. Subroutine ElResultGet (Getting Results Values at Selected Points) ....................................... 182
6.8.5. Subroutine ElInterp (Finding Element Coordinates) ................................................................ 183
6.9. Access at the Beginning and End of Various Operations ............................................................ 183
6.10. Creating Your Own Optimization Routine .................................................................................. 184
  6.10.1. Linking a Custom Optimization Program to ANSYS .............................................................. 185
  6.10.2. Subroutine userop (Defining a Custom Optimization Routine) ............................................ 185
6.11. Memory Management Routines .................................................. 188
6.11.1. Using the Memory Management Routines ................................. 188
6.11.2. Function fAnsMemAlloc (Allocating Space and Returning a Pointer) .................................................. 189
6.11.3. Subroutine fAnsMemFree (Deallocating Space) ......................... 189
6.12. Parameter Processing Routines .................................................. 190
6.12.1. Subroutine pardim (Creating a Dimensioned Parameter) .............. 190
6.12.2. Function parevl (Finding and Evaluating a Parameter) ................ 190
6.12.3. Subroutine pardef (Adding a Parameter) .................................. 190
6.13. Miscellaneous Useful Functions ................................................ 192
6.13.1. Using Function RunCommand ................................................ 192
6.13.2. Using the /UNDO Command .................................................. 192
6.13.3. Using the /HOLD command .................................................... 192

7. Accessing the ANSYS Database .................................................. 195
7.1. Inputs and Outputs for Database Access Routines ......................... 195
7.2. Types of Database Access Routines ............................................ 195
7.3. Routines for Selecting and Retrieving Nodes and Elements ............. 195
7.3.1. ndnext Function (Getting the Next Node Number) ...................... 195
7.3.2. ndprev Function (Getting the Number of the Previous Selected Node) ...................... 196
7.3.3. ndnxdf Function (Getting the Number of the Next Defined Node) .......... 196
7.3.4. ndsel Function (Selecting, Unselecting, Deleting, or Inverting a Node) .......... 196
7.3.5. elnext Function (Getting the Number of the Next Element) .......... 197
7.3.6. elprev Function (Getting the Number of the Previous Selected Element) .......... 197
7.3.7. elnxdf Function (Getting the Number of the Next Defined Element) .......... 197
7.3.8. elsel Subroutine (Selecting, Unselecting, Deleting, or Inverting an Element) .......... 197
7.4. Node Information Routines .................................................... 198
7.4.1. ndinqr Function (Getting Information About a Node) ................. 198
7.4.2. getnod Function (Getting a Nodal Point) .................................. 199
7.4.3. putnod Function (Storing a Node) .......................................... 199
7.4.4. ndgall Function (Getting the XYZ/Rotation Coordinates Vector for a Node) .......... 199
7.4.5. ndspgt Subroutine (Getting the Nodal Solution for a Node of an Element) .......... 200
7.5. Element Attribute Routines .................................................... 200
7.5.1. elmiqr Function (Getting Information About an Element) ............ 200
7.5.2. elmgft Function (Getting an Element’s Attributes and Nodes) ........ 201
7.5.3. elmput Subroutine (Storing an Element) .................................. 202
7.5.4. etyiqr Function (Getting a Data Item About an Element Type) .......... 202
7.5.5. etyget Function (Getting Information About an Element Type) ........ 203
7.5.6. etypu Subroutine (Storing Element Type Data) .......................... 203
7.5.7. echtrt Subroutine (Getting Information About Element Characteristics) .......... 203
7.5.8. etysel Subroutine (Selecting, Unselecting, Deleting, or Inverting an Element Type) .......... 204
7.5.9. mpinqr Function (Getting Information About a Material Property) .......... 204
7.5.10. mpget Function (Getting a Material Property Table) .................. 205
7.5.11. mpput Subroutine (Storing a Material Property Table) .................. 206
7.5.12. mpdel Subroutine (Deleting a Material Property Table) ................ 206
7.5.13. rlnqr Function (Getting Information About a Real Constant Set) .......... 206
7.5.14. rlgf Function (Getting Real Constant Data) ............................. 207
7.5.15. rlset Subroutine (Selecting or Deleting a Real Constant Set) .......... 207
7.5.16. csiyqr Function (Getting Information About a Coordinate System) .......... 208
7.5.17. csyget Function (Getting a Coordinate System) ........................ 208
7.5.18. csyput Subroutine (Storing a Coordinate System) ...................... 209
7.5.19. csydel Subroutine (Deleting a Coordinate System) ....................... 209
7.5.20. userac Subroutine (Demonstrates Use of Element Attribute Routines) .......... 209
7.6. Coupling and Constraint Routines ................................................................. 209
7.6.1. cpinqr Function (Getting Information About a Coupled Set) ......................... 209
7.6.2. cpget Function (Getting a Coupled Set) ..................................................... 210
7.6.3. cpput Subroutine (Storing a Coupled Set) .................................................. 210
7.6.4. cpsel Subroutine (Selecting or Deleting a Coupled Set) ............................... 211
7.6.5. ceinqr Function (Getting Information About a Constraint Equation Set) .......... 211
7.6.6. ceget Function (Getting an Constraint Equation) ....................................... 211
7.6.7. ceput Subroutine (Storing a Constraint Equation) ...................................... 212
7.6.8. cesel Subroutine (Deleting or Selecting a Constraint Equation) .................... 212

7.6.7. ceput Subroutine (Storing a Constraint Equation) ........................................ 212
7.6.8. cesel Subroutine (Deleting or Selecting a Constraint Equation) .................... 212

7.7. Nodal Loading Routines .............................................................................. 212
7.7.1. disiqr Function (Getting a Information About Constraints) ......................... 212
7.7.2. disget Function (Getting a Constraint from the Database) ......................... 213
7.7.3. disput Subroutine (Storing a Constraint at a Node) .................................... 213
7.7.4. disdel Subroutine (Deleting a Constraint at a Node) .................................. 213
7.7.5. foriqr Function (Getting Information About Nodal Loads) ........................... 214
7.7.6. forget Function (Getting a Constraint from the Database) .......................... 214
7.7.7. forput Subroutine (Storing a Nodal Load at a Node) ................................... 214
7.7.8. fordel Subroutine (Deleting a Nodal Load at a Node) ................................ 215
7.7.9. ntiqr Function (Getting Information About a Nodal Temperature) .............. 215
7.7.10. ntpget Function (Getting a Specified Nodal Temperature) ....................... 216
7.7.11. ntpput Subroutine (Storing a Nodal Temperature) .................................... 216
7.7.12. ntpdel Subroutine (Deleting a Nodal Temperature) .................................. 216
7.7.13. nhgiqr Function (Getting Information About Nodal Heat Generations) ....... 216
7.7.14. nhgget Function (Getting a Nodal Heat Generation) .................................. 217
7.7.15. nhgput Subroutine (Storing Nodal Heat Generation) .................................. 217
7.7.16. nhgdel Subroutine (Deleting a Nodal Heat Generation) ............................. 217
7.7.17. nfuiqr Function (Getting Information About Nodal Fluences) .................... 217
7.7.18. nfuget Function (Getting a Nodal Fluence) ............................................. 218
7.7.19. nfuput Subroutine (Storing a Nodal Fluence) .......................................... 218
7.7.20. nfudel Subroutine (Deleting a Nodal Fluence) ........................................ 218
7.7.21. ndciqr Function (Getting Information About Nodal Current Densities) ....... 219
7.7.22. ndcget Function (Getting a Nodal Current Density) ................................ 219
7.7.23. ndcput Subroutine (Storing a Nodal Current Density) ............................... 219
7.7.24. ndcdel Subroutine (Deleting a Nodal Current Density) ............................. 219
7.7.25. nvdciqr Function (Getting Information About Nodal Magnetic Virtual Displacements) .................. 220
7.7.26. nvdget Function (Getting a Nodal Magnetic Virtual Displacement) ............ 220
7.7.27. nvdput Subroutine (Storing a Nodal Virtual Displacement) ....................... 220
7.7.28. nvddel Subroutine (Deleting a Nodal Virtual Displacement) ...................... 220

7.8. Element Loading Routines ......................................................................... 221
7.8.1. epriqr Function (Getting Information About Element Pressure/Convection) .... 221
7.8.2. epfget Function (Getting an Element Face Pressure) ................................. 221
7.8.3. epfput Subroutine (Storing an Element Face Pressure) .............................. 222
7.8.4. epfdel Subroutine (Deleting an Element Pressure/Convection) .................... 222
7.8.5. ecviiqr Function (Getting Information About Element Convections) .......... 222
7.8.6. ecvget Function (Getting an Element Face Convection) ......................... 223
7.8.7. ecvput Subroutine (Storing an Element Face Convection) ....................... 223
7.8.8. ecvdel Subroutine (Deleting a Convection on an Element) ....................... 224
7.8.9. etpiqr Function (Getting Information About Element Temperatures) ........... 224
7.8.10. etpgget Function (Getting an Element Temperature) .............................. 225
7.8.11. etpput Subroutine (Storing an Element Temperature) ............................. 225
7.8.12. etpdel Subroutine (Deleting an Element Temperature) ............................ 225
7.8.13. ehgiqr Function (Getting Information About Element Heat Generation) .... 226
7.8.14. ehhget Function (Getting an Element Heat Generation) .......................................................... 226
7.8.15. ehhput Subroutine (Storing an Element Heat Generation) ....................................................... 227
7.8.16. ehhgdel Subroutine (Deleting an Element Heat Generation) .................................................... 227
7.8.17. efuiqr Function (Getting Information About Element Fluences) .......................................... 227
7.8.18. efuget Function (Getting an Element Fluence) ...................................................................... 228
7.8.19. efuput Subroutine (Storing an Element Fluence) ................................................................... 228
7.8.20. efudel Subroutine (Deleting an Element Fluence) .................................................................. 228
7.8.21. edcqiqr Function (Getting Information About Element Current Densities) ....................... 228
7.8.22. edcget Function (Getting Element Current Densities) ............................................................ 229
7.8.23. edcput Subroutine (Storing an Element Current Density) ....................................................... 229
7.8.24. edcdel Subroutine (Deleting an Element Current Density) ..................................................... 229
7.8.25. evdqiqr Function (Getting Information About Element Virtual Displacements) ............... 230
7.8.26. evdget Function (Getting an Element Virtual Displacement) ............................................... 230
7.8.27. evdput Subroutine (Storing an Element Virtual Displacement) ............................................. 230
7.8.28. eimiqr Function (Getting Information About Element Impedances) ................................ 231
7.8.29. eimget Function (Getting an Element Impedance) ............................................................... 231
7.8.30. eimput Subroutine (Storing an Element Impedance) ............................................................. 231
7.8.31. eimdel Subroutine (Deleting an Element Impedance) ............................................................ 232
7.8.32. esfqiqr Function (Getting Information About Element Surface Stress Data) .................. 232
7.8.33. esfget Function (Getting Element Surface Stress Data) ....................................................... 232
7.8.34. esfput Subroutine (Storing Element Surface Stress Data) ....................................................... 233
7.8.35. esfdel Subroutine (Deleting an Element's Surface Stress Data) ............................................. 233
7.8.36. esfdel Subroutine (Deleting a Flagged Surface on an Element) ............................................ 233
7.8.37. esfget Function (Getting Element Face Flagged Surfaces) .................................................. 234
7.8.38. esfqiqr Function (Getting Information About Flagged Surfaces) ......................................... 234
7.8.39. esfput Subroutine (Storing an Element Face Flagged Surface) ............................................ 234

7.9. Results Information Routines .................................................................................................. 234
7.9.1. dspiqr Function (Getting Information About Nodal Results) ............................................... 234
7.9.2. dspget Function (Getting a Nodal Result from the Database) ............................................... 235
7.9.3. dsspput Subroutine (Storing a Constraint at a Node) .............................................................. 235
7.9.4. dsspdel Subroutine (Deleting a Result at a Node) .................................................................. 235
7.9.5. emsiqr Function (Getting Information About an Element’s Miscellaneous Summable Data) ........ 236
7.9.6. emsget Function (Getting an Element’s Miscellaneous Summable Data) ............................ 236
7.9.7. emspput Subroutine (Storing an Element’s Miscellaneous Summable Data) ....................... 236
7.9.8. emspdel Subroutine (Deleting an Element’s Miscellaneous Summable Data) ..................... 237
7.9.9. enfqiqr Function (Getting Information About an Element’s Nodal Forces) ......................... 237
7.9.10. enfget Function (Getting an Element’s Nodal Forces) .......................................................... 237
7.9.11. enfput Subroutine (Storing an Element’s Nodal Forces) ....................................................... 237
7.9.12. enfdel Subroutine (Deleting an Element’s Nodal Forces) ..................................................... 238
7.9.13. ensiqr Function (Getting Information About an Element’s Nodal Stresses) ..................... 238
7.9.14. ensget Function (Getting an Element’s Nodal Stresses) ..................................................... 238
7.9.15. ensput Subroutine (Storing Nodal Stresses at an Element) .................................................... 239
7.9.16. ensdel Subroutine (Deleting an Element’s Nodal Stresses) .................................................. 240
7.9.17. engiqr Function (Getting Information About an Element’s Energies) ............................... 240
7.9.18. engget Function (Getting an Element’s Energies) ............................................................... 240
7.9.19. engput Subroutine (Storing an Element’s Energies and Volume) ........................................ 241
7.9.20. engdel Subroutine (Deleting an Element’s Energies) ............................................................ 241
7.9.21. egriqr Function (Getting Information About an Element’s Nodal Gradients) .................... 241
7.9.22. egrget Function (Getting an Element’s Nodal Gradients) ..................................................... 242
7.9.23.egrput Subroutine (Storing an Element’s Nodal Gradients) ................................................... 242
7.9.24. egrdel Subroutine (Deleting an Element’s Nodal Gradients) ................................................ 242
8. Subroutines for Users’ Convenience

8.1. Input and Output Abbreviations ............................................... ............................................... 259
8.2. General Subroutines .................................................. ................................................... .......... 259

8.9.25. eeliqr Function (Getting Information About an Element’s Nodal Elastic Strains) .............. 242
8.9.26. eelget Function (Getting an Element’s Nodal Elastic Strains) .............................................. 243
8.9.27. eelput Subroutine (Storing an Element’s Nodal Elastic Strains) ........................................ 243
8.9.28. eeldel Subroutine (Deleting an Element’s Nodal Elastic Strains) ......................................... 244
8.9.29. epliqr Function (Getting Information About an Element’s Nodal Plastic Strains) ............. 244
8.9.30. eplget Function (Getting an Element’s Nodal Plastic Strains) ............................................. 245
8.9.31. eplput Subroutine (Storing an Element’s Nodal Plastic Strains) ........................................ 245
8.9.32. epldel Subroutine (Deleting an Element’s Nodal Plastic Strains) ........................................ 246
8.9.33. ecriqr Function (Getting Information About an Element’s Nodal Creep Strains) .............. 246
8.9.34. ecrget Function (Getting an Element’s Nodal Creep Strains) .............................................. 246
8.9.35. ecrput Subroutine (Storing an Element’s Nodal Creep Strains) ........................................... 247
8.9.36. ecrdel Subroutine (Deleting an Element’s Nodal Creep Strains) ........................................ 248
8.9.37. ethiqr Function (Getting Information About an Element’s Nodal Thermal Strains) ........... 248
8.9.38. ethget Function (Getting an Element’s Nodal Thermal Stresses) ...................................... 248
8.9.39. ethput Subroutine (Storing an Element’s Nodal Thermal Stresses) .................................... 249
8.9.40. ethdel Subroutine (Deleting an Element’s Thermal, Initial, and Swelling Strains) ............... 250
8.9.41. euliq Function (Getting Information About an Element’s Euler Angles) .............................. 250
8.9.42. eulget Function (Getting an Element’s Nodal Euler Angles) ............................................... 250
8.9.43. eulput Subroutine (Storing an Element’s Euler Angles) ...................................................... 251
8.9.44. euldel Subroutine (Deleting an Element’s Euler Angles) ................................................... 251
8.9.45. efxiqr Function (Getting Information About Element Fluxes) ........................................... 251
8.9.46. efxget Function (Getting an Element Flux) ............................................................................. 252
8.9.47. efxput Subroutine (Storing an Element’s Fluxes) ................................................................. 252
8.9.48. efxdel Subroutine (Deleting Element Fluxes) .......................................................................... 252
8.9.49. elfiqr Function (Getting Information About Element Local Forces) .................................. 252
8.9.50. elfget Function (Getting an Element Local Force) ............................................................... 253
8.9.51. elfput Subroutine (Storing an Element’s Local Forces) ......................................................... 253
8.9.52. elfdel Subroutine (Deleting Element Local Forces) ............................................................... 253
8.9.53. emnq Function (Getting Information About Element Miscellaneous Non-summable Data) . ........................................................................................................................................ 254
8.9.54. emng Function (Getting an Element’s Miscellaneous Non-summable Data) ........................... 254
8.9.55. emnpm Subroutine (Storing an Element’s Miscellaneous Non-summable Data) ................. 254
8.9.56. emndel Subroutine (Deleting an Element’s Miscellaneous Non-summable Data) ................. 255
8.9.57. ecdir Function (Getting Information About Element Current Densities) ............................. 255
8.9.58. ecdir Function (Getting an Element Current Density) .......................................................... 255
8.9.59. ecdnput Subroutine (Storing an Element’s Current Densities) .............................................. 255
8.9.60. ecdnput Subroutine (Deleting Element Current Densities) .................................................. 256
8.9.61. enliqr Function (Getting Information About Element Nonlinear Tables) .......................... 256
8.9.62. enlget Function (Getting Element Nonlinear Tables) ........................................................... 256
8.9.63. enlput Subroutine (Storing an Element’s Nonlinear Tables) .................................................. 257
8.9.64. enlput Subroutine (Deleting Element Nonlinear Tables) ....................................................... 257
8.9.65. ehcieqr Function (Getting Information About Calculated Element Heat Generations) ........ 257
8.9.66. ehcget Function (Getting a Calculated Element Heat Generation) ........................................ 258
8.9.67. ehcput Subroutine (Storing an Element’s Calculated Heat Generations) ............................ 258
8.9.68. ehcdel Subroutine (Deleting Element Calculated Heat Generations) .................................... 258

8.2. General Subroutines .................................................. ................................................... .......... 259
8.2.1. dptoch Subroutine (Retrieve Eight Characters From a Double Precision Variable) ............... 259
8.2.2. wrinqr Function (Obtain Information About Output) ............................................................ 259
8.2.3. erinqr Subroutine (Obtaining Information from the Errors Common) .................................... 260
8.2.4. TrackBegin Subroutine (Beginning Tracking for a Subroutine Call) ..................................... 261
8.2.5. TrackEnd Subroutine (Ending Tracking for a Subroutine Call) ....................................................... 262
8.2.6. erhandler Subroutine (Displaying ANSYS Errors) ........................................................................... 262
8.2.7. intpr Subroutine (Doing Single Interpolation) .................................................................................. 263
8.2.8. tran3 Subroutine (Processing Geometry for 3-D Line Elements) .................................................. 263
8.2.9. systop Subroutine (Stopping an ANSYS Program Run) ................................................................. 264

8.3. Vector Functions .................................................................................................................................. 264
8.3.1. vdot Function (Computing the Dot Product of Two Vectors) ......................................................... 264
8.3.2. vsum Function (Summing Vector Components) ............................................................................... 264
8.3.3. vmax Function (Retrieving the Maximum Vector Value at a Given Location) .............................. 265
8.3.4. lastv Function (Retrieving the Position of the Last Nonzero Term in a Double Precision Vector) .................................................................................................................................................. 265
8.3.5. izero Function (Setting an Integer Vector to Zero) ........................................................................... 265
8.3.6. imove Function (Assigning Equal Values to Two Integer Vectors) ................................................. 265
8.3.7. vzero Subroutine (Initializing a Vector to Zero) ................................................................................. 265
8.3.8. vmove Subroutine (Moving One Vector into Another) ....................................................................... 266
8.3.9. vimove Subroutine (Moving One Vector into Another Incrementally) ............................................ 266
8.3.10. vinit Subroutine (Assigning a Scalar Constant to a Vector) ............................................................ 266
8.3.11. viinit Subroutine (Assigning a Scalar Constant to a Vector Incrementally) .................................... 266
8.3.12. vapb Subroutine (Setting a Vector to Sum of Two Vectors) ............................................................. 267
8.3.13. vapb1 Subroutine (Combining Two Vectors in One) ....................................................................... 267
8.3.14. vapc1 Subroutine (Multiplying a Vector to a Constant) ................................................................. 267
8.3.15. vamb Subroutine (Gets a Third Vector by Subtracting One Vector from Another) ...................... 267
8.3.16. vamb1 Subroutine (Subtracting One Vector from Another) .............................................................. 267
8.3.17. vmult Subroutine (Multiplying a Vector by a Constant) ................................................................. 268
8.3.18. vmult1 Subroutine (Multiplying a Vector by a Constant) ............................................................... 268
8.3.19. vcross Subroutine (Defining a Vector via a Cross Product) ............................................................ 268
8.3.20. vnorme Subroutine (Normalizing a Three-Component Vector) ..................................................... 268
8.3.21. vnorm Subroutine (Normalizing a Vector to Unit Length) .............................................................. 269
8.3.22. ndgxyz Function (Getting the X,Y,Z Vector for a Node) ................................................................. 269
8.3.23. ndpxyz Subroutine (Storing X,Y,Z for a Node) ................................................................................. 269

8.4. Matrix Subroutines ............................................................................................................................... 270
8.4.1. maxv Subroutine (Multiplying a Vector by a Matrix) ....................................................................... 270
8.4.2. maxv1 Subroutine (Multiplying a Vector by a Matrix) ..................................................................... 270
8.4.3. matv Subroutine (Multiplying a Vector by a Full Transposed Matrix) ............................................. 270
8.4.4. matvx1 Subroutine (Multiplying a Vector by a Full Transposed Matrix) ......................................... 271
8.4.5. matxb Subroutine (Transposing a matrix) .......................................................................................... 271
8.4.6. maat Subroutine (Changing a Matrix Value via Addition, Multiplication, and Transposition) ........ 271
8.4.7. matsym Subroutine (Filling the Upper Triangle from the Lower Triangle) ..................................... 272
8.4.8. mctac Subroutine (Transposing a symmetric matrix) .................................................................... 272
8.4.9. tran Subroutine (Transposing a matrix) ............................................................................................ 272
8.4.10. symeqn Subroutine (Solving Simultaneous Linear Equations) ...................................................... 273

A. Creating External Commands in UNIX .................................................................................................. 275
A.1. Tasks in Creating an External Command ............................................................................................ 275
A.1.1. Creating Compatible Code .............................................................................................................. 275
A.1.2. Creating a Shared Library ............................................................................................................... 276
A.1.3. Creating an External Table File ...................................................................................................... 277
A.1.4. Setting the ANSYS_EXTERNAL_PATH Environment Variable .................................................. 278
A.1.5. Using External Commands ............................................................................................................ 278
A.1.6. Checking External Command Status ............................................................................................. 278
A.1.7. Resetting External Commands ...................................................................................................... 278

B. Creating External Commands in Windows ............................................................................................. 279
B.1. Tasks in Creating an External Command ................................................................. 279
  B.1.1. Creating Compatible Code ................................................................................. 279
  B.1.2. Creating a New Project ....................................................................................... 280
  B.1.3. Creating an External Definition File ................................................................. 280
  B.1.4. Creating a Shared Library ................................................................................... 280
  B.1.5. Creating an External Table File .......................................................................... 280
  B.1.6. Setting the ANSYS_EXTERNAL_PATH Environment Variable ................................ 281
  B.1.7. Using External Commands ............................................................................... 281
  B.1.8. Checking External Command Status .................................................................... 282
  B.1.9. Resetting External Commands ......................................................................... 282

Index .................................................................................................................................... 283

List of Figures

4.1. Display Format for Z-buffered Graphics .................................................................... 90
4.2. Example Display of a Graphics File .......................................................................... 95

List of Tables

6.1. ANSYS Exit Codes ................................................................................................. 174
Preface

About the Programmer's Manual for ANSYS

The Programmer's Manual for ANSYS provides information about the various programming interfaces available to customers. This manual assumes that you have at least a basic knowledge of programming (a working knowledge of Fortran 90 would be very helpful). The two part manual includes:

Part I - Guide to Interfacing with ANSYS
   This guide describes a group of utilities as well as a set of Fortran 90 routines that you can use to directly access the ANSYS database. You can also use these capabilities to access data in any of the binary files that ANSYS writes or uses.

Part II - Guide to ANSYS User Programmable Features
   ANSYS provides a set of Fortran 90 functions and routines that are available to extend or modify the program's capabilities. Using these routines requires relinking the ANSYS program, resulting in a custom version of ANSYS. ANSYS provides an external commands capability which you can use to create shared libraries available to ANSYS (either from ANSI standard C or Fortran 90). You can use this feature to add custom extensions to ANSYS without the need to rebuild the ANSYS executable.

In addition, you can find the ANSYS Parametric Design Language Guide (APDL) as part of the ANSYS online documentation. This guide was designed for ANSYS users that have some programming skills and wish to tap the power of the ANSYS Parametric Design Language (APDL) to increase the productivity. APDL is a scripting language that is very similar to Fortran 90. The guide describes how to define parameters (variables), how to create macro programs using APDL, how to use APDL for simple user interaction, how to encrypt an APDL macro, and how to debug an APDL macro.
Part I, Guide to Interfacing with ANSYS
Chapter 1: Format of Binary Data Files

1.1. What Are ANSYS Binary Files?

The ANSYS program writes several binary files to store data during an analysis. These files are named `Job-name.ext`, where `Jobname` is the name of the analysis that caused the file to be generated and `.ext` is an extension indicating the type of data in the file. ANSYS-written binary files include the following:

- The following results files, in which the ANSYS program stores the results of solving finite element analysis problems:
  - `Jobname.RST` A structural or coupled-field analysis
  - `Jobname.RTH` A thermal analysis
  - `Jobname.RMG` A magnetic analysis
  - `Jobname.RFL` A FLOTRAN analysis
- The `Jobname.MODE` file, storing data related to a modal analysis
- The `Jobname.RDSP` file, storing data related to a reduced transient analysis.
- The `Jobname.RFRQ` file, storing data related to a reduced harmonic analysis
- The `Jobname.EMAT` file, storing data related to element matrices
- The `Jobname.SUB` file, storing data related to substructure matrices
- The `Jobname.TRI` file, storing the triangularized stiffness matrix
- The `Jobname.FULL` file, storing the full stiffness-mass matrix

The files listed above cover almost all users' needs, although there are others. For more information, see the Basic Analysis Guide.

1.1.1. Conventions Used to Describe Binary Files

In the information describing the binary file formats:

- Record ID is the identifier for this record. Not all records will have identifiers; they're indicated only for records whose record pointers are stored in a header.
- Type indicates what kind of information this record stores.
- Number of records indicates how many records of this description are found here.
- Record length indicates the number of items stored in the record.

In some record descriptions, actual variable names used may appear in the record contents area.

1.1.2. The Standard Header for ANSYS Binary Files

Each of the ANSYS program’s binary files contains a standard, 100-integer file header that describes the file contents. The header contains the items listed below, always in the order shown:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item 1</td>
<td>The file number</td>
</tr>
<tr>
<td>Item 2</td>
<td>The file format. This item has a value of 1 if the file is small format, -1 if large format.</td>
</tr>
<tr>
<td>Item 3</td>
<td>The time, in compact form (i.e., 130619 is 13:06:19)</td>
</tr>
<tr>
<td>Item</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>4</td>
<td>The date, in compact form (i.e., 20041023 is 10/23/2004)</td>
</tr>
<tr>
<td>5</td>
<td>The units of measurement used. The value of this item is as follows:</td>
</tr>
<tr>
<td></td>
<td>• 0 for user-defined units</td>
</tr>
<tr>
<td></td>
<td>• 1 for SI units</td>
</tr>
<tr>
<td></td>
<td>• 2 for CSG units</td>
</tr>
<tr>
<td></td>
<td>• 3 for U.S. Customary units (feet)</td>
</tr>
<tr>
<td></td>
<td>• 4 for U.S. Customary units (inches)</td>
</tr>
<tr>
<td>10</td>
<td>The ANSYS release level in integer form (&quot;X.X&quot; in character form)</td>
</tr>
<tr>
<td>11</td>
<td>The date of the ANSYS release</td>
</tr>
<tr>
<td>12-14</td>
<td>The machine identifier in integer form (three four-character strings)</td>
</tr>
<tr>
<td>15-16</td>
<td>The Jobname in integer form (two four-character strings)</td>
</tr>
<tr>
<td>17-18</td>
<td>The ANSYS product name in integer form (two four-character strings)</td>
</tr>
<tr>
<td>19</td>
<td>The ANSYS special version label in integer form (one four-character string)</td>
</tr>
<tr>
<td>20-22</td>
<td>The user name in integer form (three four-character strings)</td>
</tr>
<tr>
<td>23-25</td>
<td>The machine identifier in integer form (three four-character strings)</td>
</tr>
<tr>
<td>26</td>
<td>The system record size</td>
</tr>
<tr>
<td>27</td>
<td>The maximum file length</td>
</tr>
<tr>
<td>28</td>
<td>The maximum record number</td>
</tr>
<tr>
<td>31-38</td>
<td>The Jobname (eight four-character strings)</td>
</tr>
<tr>
<td>41-60</td>
<td>The main analysis title in integer form (20 four-character strings)</td>
</tr>
<tr>
<td>61-80</td>
<td>The first subtitle in integer form (20 four-character strings)</td>
</tr>
<tr>
<td>95</td>
<td>The split point of the file</td>
</tr>
<tr>
<td>97-98</td>
<td>LONGINT of filesize at write</td>
</tr>
</tbody>
</table>

### 1.2. Format of the Results File

The next few pages describe the format of the ANSYS results file. (In the following tables, records with a record ID containing an asterisk (*) are those you can read and store into the ANSYS database via the `LDREAD` command.)

Note: The pointers in the solution data headers are relative, not absolute pointers. For example, the 12th item in the solution data header will be relative to a position in the Data Set Index (ptrESL = DSI (i) + ptrESL).

This section explains the contents of the results file; that is, those files with the following extensions:

```
.rfl .brfl
.rmg .brmg
.rst .brst
.rth .brth
.lnn
```

### 1.2.1. Nomenclature

A load case contains the results for an instance in an analysis. A load case is defined by a load step number and a substep number. A load case is also categorized by a cumulative iteration number and time (or frequency) values. A load case is identified by all three methods in the results file.
The results file does not have to contain all the load cases of an analysis.

A data set is used in this chapter to designate a load case.

For a complex analysis, there will be two data sets for each load case. The first data set contains the real solution and the second contains the imaginary solution.

1.2.2. Standard ANSYS File Header

See Section 1.1.2: The Standard Header for ANSYS Binary Files for a description of this set. File number (Item 1) is 12.

1.2.3. Results File Format

*comdeck,fdresu
  c *** copyright(c) 2006 SAS IP, Inc. All rights reserved.
  c *** ansys, inc

  c ********** description of results file **********
  c --- used for the following files:
  c .rfl .brfl
  c .rnm .brnm
  c .rth .brth
  c .lnn(lxx)

  LONGINT       resufpL
  integer       resubk, resuut, resuLong, resuSpare(3)
  common /fdresu/ resufpL, resubk, resuut, resuLong, resuSpare

  c ********** common variable descriptions **********
  co resufpL   file position on file resu
  co resubk    block number for file resu (usually 6)
  co resuut    file unit for file resu (0 if not open)  FUN12
  co resuLong   0, old 32 bit integer form   1, 64 bit form (8.1)

  c See fddesc for documentation of how binary files are stored.

  c ********** file format **********
  c
  c recid tells the identifier for this record. Not all records will have
  c identifiers -- they are only indicated for those records whose
  c record pointers are stored in a header.

  c type tells what kind of information is stored in this record:
  c i - integer
  c dp - double precision
  c cmp - complex

  c nrec tells how many records of this description are found here

  c lrec tells how long the records are (how many items are stored)

  c recid type nrec lrec contents
  c --- i 1 100 standard ANSYS file header (see binhed for
       details of header contents)
  c
  c --- i 1 40 .RST FILE HEADER

  c 12, maxn, nnod, resmax, numdof,
  c maxe, nelm, kan, nsets, ptrtrend,
  c ptcrDI, ptcrTIM, ptcrLSP, ptcrELM, ptcrNOD,
  c ptrGEO, ptcrCYC, CMSFLAG, csELS, units,
  c nSector, csCord, ptrEnd8, ptrEnd8, fsiflag,
  c pmeth, noffst, eoffst, nTrans, ptrTRAN,
  c kLong, csNds, cpnrst, extopt, nlgeom,
each item in header is described below:

fun12 - unit number (resu file is 12)
maxn - maximum node number of the model
nnod - the actual number of nodes used in the solution phase
resmax - the maximum number of data sets allowed on the file (defaults to 1000; minimum allowed is 10)
numdo - number of DOFs per node
maxe - maximum element number of the finite element model
nelm - number of finite elements
kan - analysis type
nsets - number of data sets on the file
ptrend - pointer to the end of the file
ptrDSI - pointer to the data steps index table
ptrTIM - pointer to the table of time values for a load step
ptrLSP - pointer to the table of load step, substep, and cumulative iteration numbers
ptrELM - pointer to the element equivalence table
ptrNOD - pointer to the nodal equivalence table
ptrGEO - pointer to the beginning of geometry information
ptrCYC - pointer to the table of cyc sym nodal-diameters at each load step
CMSflg - CMS results flag: 0-non cms, >0-cms
units - unit system used
  - 0 - user defined units
  - 1 - SI (MKS)
  - 2 - CSG
  - 3 - U.S. Customary, using feet
  - 4 - U.S. Customary, using inches
  - 6 - MPA
  - 7 - uMKS

numdo - number of sectors for cyclic sym
csCord - Cyclic symmetry coordinate system
csEls - Cyclic sym # eles in master sector
ptrEnd8 - 23, 24 64 bit file length
fsiflag - FSI analysis flag
pmeth - p-method analysis flag
noffst - node offset used in writing file
eoffst - elem offset used in writing file
nTrans - number of SE transformation vects
ptrTRAN - pointer to SE transformation vects
kLong - 1, 64 bit integer form

--- i 1 numdo Degrees of freedom per node

DOF reference numbers are:

UX = 1, UY = 2, UZ = 3, ROTX= 4, ROTY= 5, ROTZ= 6, AX = 7, AY = 8
AZ = 9, VX =10, VY =11, VZ =12

*************** PRES=19, TEMP=20, VOLT=21, MAG =22, ENKE=23, ENDS=24
EMF =25, CURR=26, SP01=27, SP02=28, SP03=29, SP04=30, SP05=31, SP06=32
(curdof(i),i=1,numdo)

--- i 1 nnod Nodal equivalence table. This table equates the number used for storage to the actual node number (Back(i),i=1,nnod)

--- i 1 nelm Element equivalence table. The ANSYS program
stores all element data in the numerical order that the SOLUTION processor solves the elements. This table equates the order number used to the actual element number.

- **DSI** (data sets index table): This record contains the record pointers for the beginning of each data set. The first resmax records are the first 32 bits of the index, the second resmax records are the second 32 bits. To create the 64 bit pointer, use:
  
  \[
  \text{LONGPTR} = \text{longIntGet} \left( \text{first}, \text{second} \right)
  \]

  Read the solution data header as follows:

  \[
  \text{call bioBasePut} \left( \text{nblk}, \text{LONGPTR} \right)
  \]

  \[
  \text{loc} = \text{bioiqr} \left( \text{nblk}, 12 \right)
  \]

  \[
  \text{call biord} \left( \text{nblk}, \text{loc}, \ldots \right)
  \]

  The rest of the file reading continues to use the ptrXXX's that are in the headers.

- **TIM** (time/freq table): This record contains the time (or frequency) values for each data set.

- **LSP** (data set identifiers): This record contains the load step, substep, and cumulative iteration numbers for each data set.

- **TRAN** (substructure transformation vectors)

- **GEO** (geometry data header): Each item in header is described below:

  - 0 - position not used
  - maxety - the maximum element type reference number in the model
  - maxrl - the maximum real constant reference number in the model
  - ndnod - the number of defined nodes in the model
  - nel - the number of defined elements in the model
  - maxcsy - the maximum coordinate system reference number in the model
  - ptrETY - pointer to the element type index table
  - ptrREL - pointer to the real constant index table
  - ptrNOD - pointer to the nodal point locations
  - ptrCSY - pointer to the local coordinate system index table
  - ptrEID - pointer to the element index table
  - ptrMAS - pointer to the diagonal mass matrix
  - csysiz - the number of items describing a local coordinate system (usually 24)
  - elmsiz - the maximum number of nodes that a defined element may have
  - etysiz - the number of items describing an element type (=IELCSZ from echprm.inc)
  - rlsiz - the maximum number of items
defining a real constant (0, if no real constants are defined) (0)

ptrETYPL - 64 bit pointer to TYPE ptrRELL - 64 bit pointer to REAL
ptrCSYSL - 64 bit pointer to CSYS ptrNODL - 64 bit pointer to NODES
ptrEIDL - 64 bit pointer to ELEMENTS

ETY i 1 maxety The element types index table. This record contains record pointers for each element type description. (Relative to ptrETYPL for 64 bit version)

--- i numety etysiz Element type description. Each of these records is pointed to by a record pointer given in the record labeled ETY. See routines echprm and elccmt for a complete description of the items stored here.

These items are typically stored into the IELC array, and are used to determine the element type characteristics at runtime. The following items are typically of interest:

* Item 1 - element type reference number
* Item 2 - element routine number
* Items 3-14 - element type option keys (keyopts)
* Item 34 - DOF/node for this element type. This is a bit mapping of the DOF/node.
* Item 61 - number of nodes for this element type (nodelm)
* Item 63 - number of nodes per element having nodal forces, etc. (nodfor)
* Item 94 - number of nodes per element having nodal stresses, etc. (nodstr). This number is the number of corner nodes for higher-ordered elements.

REL i 1 maxrl Real constants index table. The record contains record pointers for each real constant set. (Relative to ptrRELL for 64 bit version)

--- dp numrl varies Element real constant data. These records contain real constant data used for the elements. (See the ANSYS Elements Reference manual for values for a specific element.) Each of these records is pointed to by a record pointer given in the record labeled REL. The length of these records varies for each element type (actual length is returned from routine BINRD8).

CSY i 1 maxcsy Coordinate systems index table. This record contains the record pointers for each coordinate system set. The ANSYS program writes coordinate systems only if local coordinate systems were defined. If a local system was defined, the predefined global systems 1 to 2 also will be written. The global Cartesian system 0 will never be written. (Relative to ptrCSYSL for 64 bit version)

--- dp numcsy csysiz Coordinate system description. These records contain coordinate system data for each coordinate system defined. Each of these records is pointed to by a record
The items stored in each record:

- Items 1-9 are the transformation matrix.
- Items 10-12 are the coordinate system origin (XC, YC, ZC).
- Items 13-14 are the coordinate system parameters (PAR1, PAR2).
- Items 16-18 are the angles used to define the coordinate system.
- Items 19-20 are theta and phi singularity keys.
- Item 21 is the coordinate system type (0, 1, 2, or 3).
- Item 22 is the coordinate system reference number.

The group contains the node number and coordinates (in the order Node, X, Y, Z, THXY, THYZ, THZX) for each node. (32 bit version)

(64 bit version)

Element descriptions index table. This record contains the record pointers for each element description. (LONGINT (2*nelm) for 64 bit version, relative to ptrEIDL). The order of the elements is the same as the order in the element equivalence table.

Element descriptions. Each of these records is pointed to by a record pointer given in the record labeled EID. The length of these records varies for each element (actual length is returned from routine BNRD8). nodelm shown here is the number of nodes for this element. Its value is defined in the element type description record.

The items stored in each record:

mat - material reference number

Each item is described below:

type - element type number

real - real constant reference number

secnum - section number

esys - element coordinate system

dead - death flag

= 0 - alive

= 1 - dead

solidm - solid model reference

shape - coded shape key

elnum - element number

NODES - node numbers defining the element.

(See the ANSYS Elements Reference for nodal order of an element).

Diagonal mass matrix.

The solution information is stored starting at this point in the file.

The remaining records on the file are repeated as a group nsets times (once for each data set). Item nsets is defined in the file header.
Each set of data is pointed to by a record pointer given in the record labeled DSI.

---

i 1 200 Solution data header. (was 100 in 32 bit)

pv3num, nelm, nnod, mask, itime,
iter,ncumit, nrf, cs_LSC, nmast,
ptrNSL, ptrESL, ptrRF, ptrMST, ptrBC,
rxtrap, mode, isym, kcmplx, numdof,
DOFS,
positions 51-70 - title,
positions 71-90 - stitle1,
dbmtim, dbmdat, dbfncl, soltim, soldat,
ptrOND, ptrOEL, nfldof, ptrEXA, ptrEXT
101-102 ptrEXA (was in 99)
103-104 ptrEXT (was in 100)
105-106 ptrNSL (was in 11)
107-108 ptrRF (was in 13)
109-110 ptrMST (was in 14)
111-112 ptrBC (was in 15)
113-114 ptrTRF (was in EXT 125)
115-116 ptrOND (was in 96)
117-118 ptrOEL (was in 97)
119-120 ptrESL (was in 12)
131-132 ptrVSL (was in EXT 196)
133-134 ptrASL (was in EXT 197)
139 numRotCmp
141-142 ptrRCM
143 nNodStr
145-146 ptrNODSTR

each item in header is described below:

pv3num - current solu set number
nelm - number of elements
nnod - number of nodes
mask - bitmask for the existence of several records. If a bit is set here, it indicates that the corresponding record exists on the file.
The items in the bitmask that correspond to each record are shown in the record descriptions below.
itime - loadstep
iter - iteration number
ncumit - cumulative iteration number
nrf - number of reaction forces
cs_LSC - cyclic symmetry count of the load step for this SOLVE
nmast - number of masters
ptrNSL - pointer to nodal solution
ptrESL - pointer to element solution
ptrRF - pointer to reaction forces
ptrMST - pointer to the masters
ptrBC - pointer to the boundary conditions
rxtrap - key to extrapolate integration point results to nodes
= 0 - move
= 1 - extrapolate unless active
  non-linear
= 2 - extrapolate always
mode - mode number of harmonic loading
(for cyclic symmetry: this is cs_LSF = first load step for this SOLVE)
isym - symmetry for harmonic loading
(for cyclic symmetry: this is cs_LSL = last load step for this SOLVE)
kcmplx - complex key
= 0 - real
= 1 - imaginary
numdof - number of DOFs/nodes for this data set
DOFS - DOF/node reference numbers (numdof values)
title - main title (in integer form)
stitle1 - 1st subtitle (in integer form)
dbntim - time (in compact form) when the database was last modified
dbmdat - date (in compact form) when the database was last modified
dbfncl - number of times that the database was modified
soltim - time (in compact form) when the solution for this data set was done
soldat - date (in compact form) when the solution for this data set was done
ptrOND - pointer to the ordered node list (load case files only)
ptrOEL - pointer to the ordered element list (load case files only)
nfldof - number of extra Flotran DOFs/nodes for this data set
ptrEXA - pointer to header extension for FLOTRAN DOF/extra DOF list.
ptrEXT - pointer to header extension
numRotCmp - number of rotating components
ptrRCM - pointer to RCM
nNodStr - 0, no nodal component stresses
  1, one set (TOP for shells)
  2, two sets (TOP,BOT for shells)
  3, three sets (TOP,BOT,MID)
ptrNODSTR - pointer to nodal component str
c
--- dp 1 100 --- Solution header - double precision data
timfrq, lfacto, lfactn, cptime, tref,
tunif, tbulk, volbase, tstep, 0.0,
velocity-acceleration-center of gravity terms (positions 11-28)
if pmeth=0: load data (positions 51-100)
if pmeth=1: p convergence values (positions 31-100)
c
each item is described below:
timfrq - time value (or frequency value, for a modal or harmonic analysis)
lfacto - the "old" load factor (used in ramping a load between old and new values)
lfactn - the "new" load factor
cptime - elapsed cpu time (in seconds)
tref - the reference temperature
tunif - the uniform temperature
 tbulk - Bulk temp for FLOTRAN film coefs.
 VolBase - Initial total volume for VOF
tstep - Time Step size for FLOTRAN analysis
 0.0 - position not used
c
positions 11-13 - Linear acceleration terms
positions 14-16 - Angular velocity
positions 17-19 - Angular acceleration
positions 20-22 - Angular velocity about the center of gravity
positions 23-25 - Angular acceleration about the center of gravity
positions 26-28 - Center of gravity location
Chapter 1: Format of Binary Data Files

if pmeth=1:
positions 31-100 - P convergence values

if pmeth=0:
positions 51-100 - Load data

position 53 - Convergence key (if 1, substep converged)

EXA   i   1   64
Header extension (if ptrEXA=ptrEXT, then ptrEXA is unused.)
positions 1-32 - current extra Flotran DOFs for this set
positions 33-64 - current extra Flotran DOF labels for this set

Extra Flotran DOF reference numbers are:
DENS= 1, VISC= 2, EVIS= 3, COND= 4, ECON= 5, LMD1= 6, LMD2= 7, LMD3= 8
LMD4= 9, LMD5=10, LMD6=11, EMD1=12, EMD2=13, EMD3=14, EMD4=15, EMD5=16
EMD6=17, PTOT=18, TTOT=19, PCOE=20, MACH=21, STRM=22, HFLU=23, HFLM=24
YPLU=25, TAUW=26, SPHT=27, CMUV=28
*************************** 29-32 are spares ***************************

EXT   i   1   200
Header extension
positions 1-32 - current DOF for this result set
positions 33-64 - current DOF labels for this result set
positions 65-84 - The third title, in integer form
positions 85-104 - The fourth title, in integer form
positions 105-124 - The fifth title, in integer form
position 125 - ptrTRF- pointer to FLOTRAN previous time step DOF vals
position 126 - trnvar- #dof in FLOTRAN prev time st DOF vals.
(Note 2 old steps saved, thus #DP is 2*trnvar*nNode)
position 127 - numvdof, number of velocity items per node (ANSYS transient)
position 128 - numadof, number of acceleration items per node (ANSYS transient)
position 131-133 - position of velocity in DOF record (ANSYS transient)
position 134-136 - position of acceleration in DOF record (ANSYS transient)
position 137-142 - velocity and acceleration labels (ANSYS transient)
position 143 - number of stress items (6 or 11); a -11 indicates to use principles directly and not recompute (for PSD)
position 144-146 - position of rotational velocity in DOF record (ANSYS transient)
position 147-149 - position of rotational accel. in DOF record (ANSYS transient)
position 150-155 - rotational velocity and acceleration labels (ANSYS transient)
position 160 - ptrDMI (J Integral results)
position 161 - nContours
if pmeth=1:
positions 164-200 - p convergence specs
Programmer's Manual for ANSYS, ANSYS Release 11.0. © 2003 by SAS IP, Inc. All rights reserved.
Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

1.2.3. Results File Format

- **NSL**: The DOF solution for each node in the nodal coordinate system. The DOF order is the same as shown above in the DOF number reference table. The nodal order is the same order given above in the nodal equivalence table. If a DOF for a node isn't valid, a value of 2.0**100 is used. Note 1: Sumdof = numdof + nfdof. Note 2: If, upon reading of this record, there is less than nnod*Sumdof items in the record, then only a selected set of nodes were output. Another record follows (integer, less than nnod long) which contains the list of nodes for which DOF solutions are available. (bit 10 (PDBN) in mask)

- **VSL**: The velocity solution for each node in the nodal coordinate system. The description for the DOF solution above also applies here. ANSYS transient. (bit 27 (PDVEL) in mask)

- **ASL**: The acceleration solution for each node in the nodal coordinate system. The description for the DOF solution above also applies here. ANSYS transient. (bit 28 (PDACC) in mask)

- **RF**: Reaction force DOFs. This index is calculated as (N-1)*numdof+DOF, where N is the position number of the node in the nodal equivalence table, and DOF is the DOF reference number. (bit 11 (PDBR) in mask)

- **MST**: Master DOF list. This index is calculated as (N-1)*numdof+DOF, where N is the position number of the node in the nodal equivalence table, and DOF is the DOF reference number. (bit 4 in mask)

- **BC**: Boundary condition index table. (bit 23 (PDBBC) in mask)

Each item is described below:

- **numdis**: number of nodal constraints
- **ptrDIX**: pointer to the table of nodes having nodal constraints
- **ptrDIS**: pointer to nodal constraint values
- **numfor**: number of nodal input force loadings
- **ptrFOR**: pointer to the table of nodes having nodal forces
- **ptrFIX**: pointer to nodal force values

Positions 7-40 are unused.
Chapter 1: Format of Binary Data Files

- **DIX** i 1 1 numdis: Nodal constraint DOF. This index is calculated as N*32+DOF, where N is the node number and DOF is the DOF reference number. Values are in the same order as the DOF number reference table.

- **DIS** dp 1 4*numdis: Nodal constraints. This record contains present and previous values (real and imaginary) of the nodal constraints at each DOF.

- **FIX** i 1 numfor: Nodal input force DOFs. This index is calculated as N*32+DOF, where N is the node number and DOF is the DOF reference number. Values are in the same order as the DOF number reference table.

- **FOR** dp 1 4*numfor: Nodal forces. This record contains present and previous values (real and imaginary) of the nodal input force loadings at each DOF.

- **TRF** dp 1 28*nnod: Two displacement result sets for transient solution in FLOTRAN (bit 24 (PDTRFL) in mask)

- **OND** i 1 nnod: Ordered node list. This record exists for a load case file only.

- **OEL** i 1 nelm: Ordered element list. This record exists for a load case file only.

- **ESL** i 1 2*nelm: Element solutions index table. This record contains pointers to each element solution. The order of the elements is the same as the order in the element equivalence table. (bit 12 (PDBE) in mask)

- **RCM** dp 1 6*numRotCmp: Angular velocities (3) and angular accelerations (3) of components.

- **DMI** dp 1 3+nContours: Crack ID, Contour ID, TipNode, J Integral values

The solution information for each individual element is stored starting at this point in the file. The next 23 records on the file are repeated as a group nelm times (once for each element). Item nelm is defined in the file header.

--- i 1 25: Individual element index table.

- ptrEMS - pointer to misc. data
- ptrENF - pointer to nodal forces
- ptrENS - pointer to nodal stresses
- ptrENG - pointer to volume and energies
- ptrEGR - pointer to nodal gradients
- ptrEEL - pointer to elastic strains
- ptrEPL - pointer to plastic strains
- ptrECR - pointer to creep strains
- ptrETH - pointer to thermal strains
- ptrEUl - pointer to euler angles
- ptrEFX - pointer to nodal fluxes

(Relative to ptrESL for 64 bit version)
c                                    ptrELF - pointer to local forces
c                                    ptrEMN - pointer to misc. non-sum values
c                                    ptrECD - pointer to element current
c                                             densities
c                                    ptrENL - pointer to nodal nonlinear data
c                                    ptrEHC - pointer to calculated heat
c                                             generations
c                                    ptrEPT - pointer to element temperatures
c                                    ptrESF - pointer to element surface
c                                             stresses
c                                    ptrETB - pointer to ETABLE items(post1 only
c                                    ptrECT - pointer to contact data
c                                    ptrEXY - pointer to integration point
c                                             locations
c                                    ptrEBA - pointer to back stresses
c                                    ptrESV - pointer to state variables
c                                    0      - position not used

Note! If ptrXXX is negative, then all
|ptrXXX| items are zero and are not on
the file.

c                                    EMS dp  1  varies
Element summable miscellaneous data. The
contents and number of data items is
element-dependent. For a list of what's
available, see the SMISC item in the
description of the ETABLE command in the
ANSYS Commands Reference.

c                                    ENF dp  1  varies
Element nodal forces. This record contains
the forces at each node, in the same DOF
order as the DOF number reference table.
For static, damping, and inertia forces, a
set of forces will be repeated (as
appropriate). Number of data items stored
in this record can be calculated as
follows: nodfor*NDOF*M, where NDOF is the
number of DOFs/node for this element,
nodfor is the number of nodes per element
having nodal forces (defined in element
type description record), and M may be 1,
2, or 3. For a static analysis, M=1 only.
For a transient analysis, M can be 1, 2,
or 3.

c                                    ENS dp  1  varies
Element nodal component stresses. This
record contains the stresses at each corner
node, in the order SX,SY,SZ,SXY,SYZ,SXZ,S1,
S2,S3,S1,SIGE. Nodal order corresponds to
the connectivity defined in the element
description. Stresses can be nodal values
extrapolated from the integration points or
values at the integration points moved to
the nodes. If an element is nonlinear,
integration point values always will be
written. (See item rxtrap in the solution
header for the setting.) An element is
considered nonlinear when either plastic,
creep, or swelling strains are present.

Definition of common terms referred here
and in subsequent EEL, EPL, ECR, ETH,
ENL, EUL and EPT sections:

c                                    nodstr - number of nodes per element
having stresses, strains, etc.
For higher-order elements, nodstr
equals to the number of corner
nodes (e.g., for 20-noded SOLID186,
nodstr = 8).
c
nodfor - number of nodes per element
having nodal forces, etc.
Chapter 1: Format of Binary Data Files

ENG dp 1 11 Element volume and energies.

volume, senergy, aenergy, kenergy, coenergy, incenergy, 0.0, 0.0, thenergy, 0.0, 0.0

each item is described below:

volume - element volume
senergy - element energy associated with the stiffness matrix
aenergy - artificial hourglass energy
kenergy - kinetic energy
coenergy - co-energy (magnetics)
incenergy - incremental energy (magnetics)
0.0 - position not used
0.0 - position not used
thenergy - thermal dissipation energy
Element nodal field gradients. This record contains the gradients at each corner node in the order X,Y,Z. Nodal order corresponds to the connectivity defined in the element description. If this is a coupled-field analysis, the data is stored in the following order (as available): fluid, thermal (TEMP), electric (VOLT), and magnetic (AZ). Gradients can be nodal values extrapolated from the integration points or values at the integration points moved to the nodes. See item rxtrap in the solution header for the setting. The number of items in this record is \( \text{nodstr} \times 3 \times N \), where \( N \) can be 1, 2, 3, or 4 (depending on the coupled-field conditions).

NOTE: \text{nodstr} is defined in the element type description record.

Element nodal component elastic strains. This record contains strains in the order X,Y,Z,XY,YZ,XZ,EQV. Elastic strains can be nodal values extrapolated from the integration points or values at the integration points moved to the nodes. If an element is nonlinear, integration point values always will be written. See item rxtrap in the solution header for the setting. An element is considered nonlinear when either plastic, creep, or swelling strains are present. For beam elements, see item LEPL in the Output Data section for the particular element in the ANSYS Elements Reference.

NOTE: See ENS record section for more details on record content and length.

Element nodal component plastic strains. This record contains strains in the order X,Y,Z,XY,YZ,XZ,EQV. Plastic strains are always values at the integration points moved to the nodes. For beam elements, see item LEPPL in the Output Data section for the particular element in the ANSYS Elements Reference.

NOTE: See ENS record section for more details on record content and length.

Element nodal component creep strains. This record contains strains in the order X,Y,Z,XY,YZ,XZ,EQV. Creep strains are always values at the integration points moved to the nodes. For beam elements, see item LEPCR in the Output Data section for the particular element in the ANSYS Elements Reference.

NOTE: See ENS record section for more details on record content and length.
Element nodal component thermal strains. This record contains strains in the order $X, Y, Z, XY, YZ, XZ, EQV$ plus the element swelling strain. Thermal strains can be nodal values extrapolated from the integration points or values at the integration points moved to the nodes. If the element is nonlinear, integration point data always will be written. (An element is considered nonlinear when either plastic, creep, or swelling strains are present.) See item rxtrap in the solution header for the setting. For beam elements, see item LEPTH in the description of the Output Data section for the particular element in the ANSYS Elements Reference.

NOTE: See ENS record section for more details on record content and length.

Element Euler angles. This record contains the Euler angles ($THXY, THYZ, THZX$).

* For lower-order elements, angles are at the centroid and the number of items in this record is 3.
* For higher-order elements, angles are at each corner node and the number of items in this record is nodstr*3.
* For layered shells, higher-order layered solid elements, and layered SOLSH190 and SOLID185, angles are at each corner node, plus the layer orientation angle for each layer. The number of items in this record is (nodstr*3)+NL.
* For other lower-order layered solid elements, Euler angles are at the centroid, plus the layer orientation angle for each layer. Therefore, the number of items in this record is 3 + NL.

NOTE: See ENS record section for definition of terms NL and nodstr.

Element nodal field fluxes. This record contains the fluxes at each corner node in the order $X, Y, Z$. If this is a coupled-field analysis, the flux data is stored in the following order: thermal, electric, magnetic. Nodal order corresponds to the connectivity defined in the element description. Fluxes can be nodal values extrapolated from the integration points or values at the integration points moved to the nodes. See item rxtrap in the solution header for the setting. The number of items in this record is nodstr*3*N, where N can be 1, 2, or 3 depending on the coupled-field conditions.

NOTE: nodstr is defined in the element type description record.

Element nodal coupled-field forces. This record lists the forces at each node in the order $X, Y, Z$. For most elements, the number of items in this record is nodfor*3. However, for the PLANES3 element, the number of items in this record is either nodfor*3 or nodstr*3. (See the description
of KEYOPT(7) for PLANE53 in the ANSYS Elements Reference.) NOTE: nodfor and nodstr are defined in the element type description record.

NOTE: nodstr is defined in the element type description record.

Element nonsummable miscellaneous data. The contents and number data items for this record is element-dependent. See the description for item NMISC of the ETABLE command in the ANSYS Commands Reference.

Element current densities. This record contains the calculated current densities in the order X,Y,Z.

Element nodal nonlinear data. This record stores nonlinear data at each corner node in the order SEPL, SRAT, HPRES, EPEQ, PSV, PLWK, CRWK, and ELENG followed by 2 spares.

Each item is described below:
- SEPL - equivalent stress parameter
- SRAT - stress ratio
- HPRES - hydrostatic pressure
- EPEQ - accumulated equivalent plastic strain
- PSV - plastic state variable
- PLWK - plastic strain energy density (work)
- CRWK - creep strain energy density (work)
- ELENG - elastic strain energy density

* See ENS record section for details on solid and shell elements.
* For beam elements, the contents and number of data items in this record is element-dependent. See the description of item NLIN in the Output Data section for the particular element in the ANSYS Elements Reference.

Element heat generation. This record stores the calculated heat generation.

Element structural nodal temperatures.

* For solid elements and SHELL41, the record contains nodal temperatures at each node and the number of items in this record is nodfor.
* For shell elements, except SHELL41 and SHELL91, the record contains nodal temperatures at each corner node for the top surface and the bottom surface. The number of items in this record is nodstr*2.
* For SHELL91 and SOLID191, the record contains nodal temperatures at each corner node for the bottom of the bottom layer, and each succeeding interlayer surface up to the top of the top layer. The number of items in this record is (NL+1)*nodstr.
* For layered shell elements SHELL181, SHELL281, SHELL208, SHELL209, and layered solid elements SOLID185, SOLID186, and SOLSH190, the record contains temperatures for each layer at each corner node (first at the bottom layer surface, then the top). Therefore, the number
of items in this record is NL*2*nodstr for layered shells and NL*nodstr for layered solid elements.

* For layered membrane elements (SHELL181, SHELL281, SHELL208, and SHELL209 with KEYOPT(1)=1), the record contains temperatures for each layer at each corner node. Therefore, the number of items in this record is NL*nodstr.

* For beam elements, the contents and number of data items in this record is element-dependent. See the description of item LBFE in the Output Data section for the particular element in the ANSYS Elements Reference.

NOTE: See ENS record section for definition of terms NL, nodstr, and nodfor.

ESF dp 1 nsurf*19 Element surface stresses. The length of this record is nsurf*19 where nsurf is the number of surfaces that have surface stress information. The stress information is simply repeated in the format shown below for each surface.

* For 2d elements:

facenm, area, temp, press, eppar, epper, epz, 0.0d0, spar, sper, sz, 0.0d0, 0.0d0, 0.0d0, s1, s2, s3, sint, seqv

* For 3d elements:

facenm, area, temp, press, epx, epy, epz, epxy, sx, sy, sz, sxy, 0.0d0, 0.0d0, s1, s2, s3, sint, seqv

* For axisymmetric elements:

facenm, area, temp, press, eppar, epper, epz, epsh, spar, sper, sz, 0.0d0, 0.0d0, ssh, s1, s2, s3, sint, seqv

Each item is described below:

facenm - face number
area - face area
temp - face temperature
press - face pressure
epx - strain parallel to face
epy - strain parallel to face
epz - strain perpendicular to face
epsh - shear strain
eppar - strain parallel to face
epper - strain perpendicular to face
epsh - torsion shear strain
sx - stress parallel to face
sy - stress parallel to face
sz - stress perpendicular to face
sxy - shear stress
spar - stress parallel to face
sper - stress perpendicular to face
ssh - torsion shear stress
s1 - S(1)
s2 - S(2)
s3 - S(3)
sint - S(INT)
seqv - S(EQV)
1.3. Description of the Reduced Displacement File

This section explains the content of the reduced displacement file (jobname.rdsp).

1.3.1. Standard ANSYS File Header

See Section 1.1.2: The Standard Header for ANSYS Binary Files for a description of this set. File number (Item 1) is 10.

1.3.2. RDSP File Format

*comdeck,fdrdsp

** *** copyright(c) 2006 SAS IP, Inc. All rights reserved.
** *** ansys, inc

** ********** description of reduced displacement file **********
character*8 RDSPNM parameter (RDSPNM='rdsp ')
LONGINT rdspfpL, rdspfp
integer rdspbk, rdsput
common /fdrdsp/ rdspfpL, rdspbk, rdsput
 equivalence (rdspfp,rdspfpL)

** write: lnfrcl,lnfrin,lnfrwr
** write: rdtrcl,rdtrin,rdtrwr
** read: rdrtrrs,rdrtrs

** ********** common variable descriptions **********
co rdspfpL file position on file rdsp
c0 rdspbk block number for file rdsp
co rdspit   file unit for file rdsp

See fddesc for documentation of how binary files are stored.

********** file format **********

recid tells the identifier for this record. Not all records will have
identifiers -- they are only indicated for those records whose
record pointers are stored in the second file header.

type tells what kind of information is stored in this record:
   i - integer
dp - double precision
cmp - complex

nrec tells how many records of this description are found here
lrec tells how long the records are (how many items are stored)

recid  type  nrec  lrec     contents
---    i     1    100  standard ANSYS file header (see binhed for
details of header contents)
---    i     1     40  .RDSP FILE HEADER
   fun10, nmrow, nmatrix, nmode, numdof,
   maxn, wmax, lenbac, ngaps, ncumit,
   kan, nres, ndva, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0,
   ptrDOF, ptrDNC, ptrSTF, ptrMAS, ptrDMP,
   ptrFRQ, ptrDSP, ptrSTFh, ptrMASh, ptrDMPh,
   ptrFRQh, ptrDSPh, ptrDVA, ptrDVAh, 0,
   0, 0, 0, 0, 0

each item in header is described below:

fun10 - unit number (rdsp file is 10)
nmrow - number of rows/columns in matrices
nmatrix - number of reduced matrices on the
           file
nmode - number of modes extracted during
        modal analysis (or nmrow if reduced
        method)
numdof - number of dofs per node
maxn - maximum node number
wmax - maximum wavefront
lenbac - number of nodes
ngaps - number of gaps
ncumit - total number of iterations done
          during analysis
kan - analysis type
      = 5 for reduced transient analysis
nres - number of residual vectors used
ndva - length of DVA
0 - position not used
0 - position not used
0 - position not used
0 - position not used
0 - position not used
0 - position not used
ptrDOF - pointer to degree of freedom set
ptrDNC - pointer to nodal constraints
ptrSTF - pointer to the reduced stiffness
ptrMAS - pointer to the reduced mass matrix
ptrDMP - pointer to the reduced damping
matrix or mode shapes
ptrFRQ - pointer to the frequencies
ptrDSP - pointer to the calculated
          displacements
ptrSTFh - High part of reduced stiffness ptr
ptrMASh- High part of reduced mass ptr
ptrDMPh- High part of reduced damping ptr
ptrFRQh- High part of frequency ptr
ptrDSPh- High part of displacement ptr
ptrDVA - pointer to modal disp, velo and acc
ptrDVAh- High part of modal disp, velo and acc
0      - position not used
0      - position not used
0      - position not used
0      - position not used
0      - position not used
0      - position not used
0      - position not used
0      - position not used
0      - position not used
numdof Degrees of freedom per node
(curdof(i),i=1,numdof)
dof reference numbers are:
UX  = 1, UY  = 2, UZ  = 3, ROTX= 4, ROTY= 5, ROTZ= 6, AX = 7, AX = 8
AZ = 9, VX =10, VY =11, VZ =12 ****** 13-18 are spares ********
EMF =25, CURR=26 ****** 27-32 are spares ******************
lenbac This table equates the actual node number to
the number used for storage.
(Back(i),i=1,lenbac)
dftime, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, timend
each item is described below:
dftime - the time increment
0.0      - position not used
0.0      - position not used
0.0      - position not used
0.0      - position not used
0.0      - position not used
0.0      - position not used
0.0      - position not used
timend - the final time of the analysis
nmrow Degree of freedom set used
The DOFs are calculated as (N-1)*numdof+DOF,
where N is the position number of the node in
the nodal equivalence table and DOF is the
DOF reference number given above.
If the analysis uses the reduced method, the
original DOF order (see next record) is
rearranged so that DOFs having nodal
constraints are listed first.
If the analysis uses the mode superposition
method (using the reduced mode extraction
technique), the DOF order is the same as the
original order (see next record).
(l(i),i=1,nmrow)
nmrow+1 Original reduced set of DOFs used.
The DOFs are calculated as (N-1)*numdof+DOF,
where N is the position number of the node in
the nodal equivalence table and DOF is the
DOF reference number given above.
If the analysis uses the reduced method, the
original DOF order, plus the number of nodal
constraints (nbcdsp), is stored.
If the analysis uses the mode superposition
method (using the reduced mode extraction
technique), this record matches the previous
The nmrow+1 entry will be zero.

(i-1,nmrow),nbcdsp

This record is present only if the analysis uses the reduced method and nbcdsp > 0 (see record at ptrDOF). These numbers are the positions in the previous record of dofs with a nodal constraint. These are nodal constraints only on nodes that also are masters.

(lorig(i),i-1,nmrow),nbcdsp

Reduced stiffness matrix. Each row of the matrix is stored as a record. The matrix is present only if nmatrx > 0 and analysis is not using mode superposition method (using the subspace mode extraction method). Row order is the same as the DOF order in record at ptrDOF.

Reduced mass matrix. Each row of the matrix is stored as a record. The matrix is present only if nmatrx > 1 and analysis is not using mode superposition method (using the subspace extraction technique). Row order is the same as the DOF order in record at ptrDOF.

Reduced damping matrix or mode shapes. If the analysis uses the reduced method, each record will be nmrow items in length. The reduced damping matrix is present only if nmatrx > 2. There will be nmrow records of this type stored here. Row order is the same as the DOF order in record at ptrDOF.

If the analysis uses the mode superposition method (using the reduced mode extraction technique), each record will be nmode items in length. These records contain mode shapes (eigenvectors) of the frequencies (eigenvalues) actually used in the harmonic analysis. There will be nmode records of this type stored here, with the first N records containing the mode shapes and the other records containing zeros, where N is the number of modes actually used in the harmonic analysis. Order corresponds to the DOF order given in record at ptrDOF.

If the analysis uses the mode superposition method (using the subspace mode extraction technique), this record will not be present.

Frequencies extracted from the modal analysis. This record is present only if the analysis uses the mode superposition method. The first nmode values are the frequencies extracted from the modal analysis. The remaining values have no meaning.

The number of iterations is stored as ncumit. (see above records that deal with time)

The first nmrow entries are the displacements in the same order as the original set of DOFs.
1.4. Description of the Reduced Complex Displacement File

This section explains the content of the reduced complex displacement file (jobname.rfrq).

1.4.1. Standard ANSYS File Header

See Section 1.1.2: The Standard Header for ANSYS Binary Files for a description of this set. File number (Item 1) is 10.

1.4.2. RFRQ File Format

*comdeck,fdrfrq

---

```
c$ comment (see record AFTER ptrDOF). For the last five
centries:
1. Time for these displacements
2. Load step number
3. Substep number
4. Cumulative iteration number
5. Scale factor (zero if the analysis uses
the reduced method).

(u(i),i=1,nmrow),time,itime,iter,ncumit,
scale

Note: If, upon reading of this record, there
is less than nmrow+5 items in the record,
then only a selected set of nodes were
output. Another record follows (integer, less
than lenbac long) which contains the list of
nodes for which DOF solutions are available.

---

--- dp 1 ngaps
Gap restoring forces. The order of these
forces corresponds to the node position order
given in record at ptrDNC. This record is
present only if ngaps > 0.

(fgaps(i),i=1,ngaps)
```

---

```
c *** The next 3 records are kept for possible restart using mode superposition
method. They are overwritten upon restarting. They are written once (last
loadstep).

--- dp 1 ndva+5
Calculated modal displacements
The first ndva entries are the modal
displacements. For the last five
entries:
1. Time for these displacements
2. Load step number
3. Substep number
4. Cumulative iteration number
5. Scale factor (zero if the analysis uses
the reduced method).

(um(i),i=1,ndva),time,itime,iter,ncumit,
scale
```

---

```
c ***
c
--- dp 1 ndva
Calculated modal velocities
(vm(i),i=1,ndva)
```

---

```
c ***
c
--- dp 1 ndva
Calculated modal accelerations
(am(i),i=1,ndva)
```

---

```
c
**********
description of reduced complex displacement file **********
character*8  RFRQNM
parameter (RFRQNM='rfrq    ')
LONGINT  rfrqfpL, rfrqfp
integer  rfrqbk, rfrqut
```

---

Programmer's Manual for ANSYS . ANSYS Release 11.0 . 002328 . © SAS IP, Inc. All rights reserved.
Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.
Chapter 1: Format of Binary Data Files

common /fdrfq/ rfrqfpL, rfrqbk, rfrqut
equivalence (rfrqfp,rfrqfpL)

c write: harmcl,harmin,harmwr
c write: hrfrcl,hrfreq
c read: harstr

************ common variable descriptions ************
co rfrqfpL      file position on file rfrq
co rfrqbk       block number for file rfrq
co rfrqut       file unit for file rfrq

c See fddesc for documentation of how binary files are stored.

c ********** file format **********
recid tells the identifier for this record. Not all records will have
two records will have

identifiers -- they are only indicated for those records whose
record pointers are stored in the second file header.

type tells what kind of information is stored in this record:
i  - integer
dp - double precision
cmp - complex

nrec tells how many records of this description are found here

lrec tells how long the records are (how many items are stored)

c recid type nrec lrec contents

c --- i 1 100 standard ANSYS file header (see binhed for
details of header contents)
c

c --- i 1 40 .RFRQ FILE HEADER

c
fun10, nmrow, nmatrx, nmode, numdof,
maxn, wffmax, lenbac, 0, ncumit,
kan, 0, 0, 0, 0,
0, 0, 0, 0, 0,
ptrDOF, ptrDNC, ptrSTF, ptrMAS, ptrDMP,
ptrFRQ, ptrDSP, ptrSTFh, ptrMASH, ptrDMPh,
ptrFRQh, ptrDSPh, 0, 0, 0, 0,
0, 0, 0, 0, 0

each item in header is described below:

fun10  - unit number (rfrq file is 10)
nmrow - number of rows/columns in matrices
nmatrx - number of reduced matrices on file
nmode  - number of modes extracted during
modal analysis (or nmrow if reduced
method)
numdof - number of dofs per node
maxn  - maximum node number
wffmax - maximum wavefront
lenbac - number of nodes
0  - position not used
ncumit - total number of iterations done
during analysis
kan  - analysis type
    = 6  - reduced harmonic
0  - position not used
0  - position not used
0  - position not used
0  - position not used
0  - position not used
0  - position not used
ptrDOF - pointer to degree of freedom set
c used in model
ptrDNC - pointer to nodal constraints
ptrSTF - pointer to the reduced stiffness matrix
ptrMAS - pointer to the reduced mass matrix
ptrDMP - pointer to the reduced damping matrix or mode shapes
ptrFRQ - pointer to the frequencies
ptrDSP - pointer to the calculated displacements
ptrSTFh- High part of STF pointer
ptrMASh- High part of MAS pointer
ptrDMPH- High part of DMP pointer
ptrFRQh- High part of FRQ pointer
ptrDSPH- High part of DSP pointer
0 - position not used
0 - position not used
0 - position not used
0 - position not used
0 - position not used
0 - position not used
0 - position not used
0 - position not used
0 - position not used
0 - position not used
0 - position not used

--- i 1 numdof Degrees of freedom per node
        (curdof(i),i=1,numdof)
dof reference numbers are:
UX = 1, UY = 2, UZ = 3, ROTX= 4, ROTY= 5, ROTZ= 6, AX = 7, AY = 8
AZ = 9, VX =10, VY =11, VZ =12 ****** 13-18 are spares ********
******* PRES=19, TEMP=20, VOLT=21, MAG =22, ENKE=23, ENDS=24
EMF =25, CURR=26 ******* 27-32 are spares ****************

--- i 1 lenbac This table equates the actual node number to
the number used for storage.
        (Back(i),i=1,lenbac)

--- dp 1 10 Unused record. contents:
1.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0

DOF i 1 nmrow Degree of freedom set used
        The DOFs are calculated as (N-1)*numdof+DOF,
        where N is the position number of the node in
        the nodal equivalence table and DOF is the
        DOF reference number given above.

        If the analysis uses the reduced method, the
        original DOF order (see next record) is
        rearranged so that DOFs having nodal
        constraints are listed first.

        If the analysis uses the mode superposition
        method (using the reduced mode extraction
        technique), the DOF order is the same as the
        original order (see next record).
        (l(i),i=1,nmrow)

--- i 1 nmrow+1 Original reduced set of DOFs used.
        The DOFs are calculated as (N-1)*numdof+DOF,
        where N is the position number of the node in
        the nodal equivalence table and DOF is the
        DOF reference number given above.
        If the analysis uses the reduced method, the
        original DOF order, plus the number of nodal
        constraints (nbcdsp), is stored.

        If the analysis uses the mode superposition
        method (using the reduced mode extraction
        technique), this record matches the previous
        record. The nmrow+1 entry will be zero.
        (lorig(i),i=1,nmrow),nbcdsp
c DNC      i   1   nbcdsp  This record is present only if the analysis
    uses the reduced method and nbcdsp > 0 (see
    record at ptrDOF). These numbers are the
    positions in the previous record of dofs with
    a nodal constraint. These are nodal
    constraints only on nodes that also are
    masters.  
    (na(i),i=1,nbcdsp)

c STF      dp   nmrow  nmrow  Reduced stiffness matrix. Each row of the
    matrix is stored as a record. The matrix is
    present only if nmatrx > 0 and analysis is
    not using mode superposition method (using
    the subspace mode extraction method). Row
    order is the same as the DOF order in record
    at ptrDOF.  
    (ak(i,j),i=1,nmrow)

c MAS      dp   nmrow  nmrow  Reduced mass matrix. Each row of the matrix
    is stored as a record. The matrix is present
    only if nmatrx > 1 and analysis is not using
    mode superposition method (using the subspace
    extraction technique). Row order is the same
    as the DOF order in record at ptrDOF.  
    (am(i,j),i=1,nmrow)

c DMP      dp   varies  varies  Reduced damping matrix or mode shapes.  
    If the analysis uses the reduced method,
    each record will be nmrow items in length.  
    The reduced damping matrix is present only
    if nmatrx > 2. There will be nmrow records of
    this type stored here. Row order is the same
    as the DOF order in record at ptrDOF.  
    If the analysis uses the mode superposition
    method (using the reduced mode extraction
    technique), each record will be nmode items
    in length. These records contain mode shapes
    (eigenvectors) of the frequencies
    (eigenvalues) actually used in the harmonic
    analysis. There will be nmode records of this
    type stored here, with the first N records
    containing the mode shapes and the other
    records containing zeros, where N is the
    number of modes actually used in the harmonic
    analysis. Order corresponds to the DOF order
    given in record at ptrDOF.  
    If the analysis uses the mode superposition
    method (using the subspace mode extraction
    technique), this record will not be present.  
    (psi(i,j),i=1,nmrow) (or ac)

c FRQ      dp   1   nmrow  Frequencies extracted from the modal analysis.  
    This record is present only for analyses using
    the mode superposition method (using the
    reduced mode extraction technique).  
    (freq(i),i=1,nmrow)

c DSP      cmp  ncumit  nmrow+5  Calculated complex displacements
    The first nmrow entries are the displacements
    in the same order as the original set of DOFs
    (see record AFTER ptrDOF). For the last five
    entries:   
    Real part  Imag part
    1. frequency for these
    frequency increment
    values
    2. load step number
    substep number
    3. cumulative iteration
    number

1.5. Description of the Modal Results File

This section explains the content of the modal results file (jobname.mode).

1.5.1. Standard ANSYS File Header

See Section 1.1.2: The Standard Header for ANSYS Binary Files for a description of this set. File number (Item 1) is 9.

1.5.2. MODE File Format

*comdeck,fdmode
co copyright(c) 2006 SAS IP, Inc. All rights reserved.
c ansys, inc.

co ********** description of modal result file **********

c mpg fdmode < modspc romstr lire_freq_mode lire_nb_mode: mode file desc

c character*8 MODENM
co parameter (MODENM='mode ')
co LONGINT modefpL, modefp
co integer modebk, modeut

co ********** common variable descriptions **********
co modefpL file position on file mode
co modebk block number for file mode
co modeut file unit for file mode

co See fddesc for documentation of how binary files are stored.

c ********** file format **********

co recid tells the identifier for this record. Not all records will have
co identifiers -- they are only indicated for those records whose
co record pointers are stored in the second file header.

c type tells what kind of information is stored in this record:
co i - integer
co dp - double precision
co cmp - complex

c nrec tells how many records of this description are found here

c lrec tells how long the records are (how many items are stored)

co recid  type  nrec  lrec  contents
co ---  i  1  100  standard ANSYS file header (see binhed for
Chapter 1: Format of Binary Data Files

details of header contents)

---

.fun09, nmrow, nmatrix, nmode, numdof,
maxn, wfmax, lenbac, 0, nontp,
lumpms, extopt, SvCode, kan, ldstep,
numitr, expbeg, expend, nspect, nSPdat,
ptrRDF, ptrFRQ, ptrPRT, ptrSHP, ptrLOD,
ptrSTF, ptrMAS, ptrDMP, ptrCOF, ptrDCF,
ptrLPM, ptrSPI, ptrSPH, ptrLDP, ptrSTFH,
ptrMASH, ptrDMPH, ptrLPMH, ptrSPIH, ptrIRSH1,
ptrIRSHh, PowerDyn, 0, 0, 0,
0, 0, 0, 0, 0,
0, 0, 0, 0, 0,
0, 0, 0, 0, 0

each item in header is described below:

fun09 - unit number (mode file is 9)
nmrow - number of rows/columns in matrices
(maxn*numdof). If extopt = 0, nmrow
is the number of rows in the
reduced matrices and the number of
master degrees of freedom.
nmatrix - number of reduced matrices on the
file (applies only if extopt=0)
nmode - number of modes extracted
numdof - number of dof per node
maxn - maximum node number (If extopt = 3
or 4, the actual number of nodes is
referenced.)
wfmax - maximum wavefront (Does not apply
if extopt = 3 or 4.)
lenbac - number of nodes
0 - position not used
nontp - number of equations on the .TRI
file (Does not apply if extopt = 0.)
lumpms - lumped mass key
= 0 - default matrix type
= 1 - lumped
(Does not apply if extopt = 3 or
4.)
extopt - mode extraction method
= 0 - reduced
= 1 - subspace
= 3 - unsymmetric Lanczos
= 4 - damped Lanczos
= 6 - block Lanczos
= 7 - QR damped
= 8 - AMLS
SvCode - Solver assembly code
= 0 Frontal assembly (SV_ANSYS)
= 1 Symbolic assembly (SV_CASI)
kan - analysis type
= 1 - buckling
= 2 - modal
ldstep - load step number
numitr - total number of cumulative
iterations done during analysis
(Does not apply if extopt = 3 or
4.)
expbeg - beginning of the frequency range of
interest
expend - end of the frequency range of
interest
nspect - number of spectra
nSPdat - number of data items per spectrum
ptrRDF - pointer to reduced degree of
freedom set used in model
ptrFRQ - pointer to the frequencies
# 1.5.2. MODE File Format

In this section, we will discuss the file format for the MODE file in ANSYS. The MODE file is used to store modal analysis results, including participation factors, mode shapes, frequencies, and modal damping coefficients.

- **ptrPRT**: Pointer to the participation factors.
- **ptrSHP**: Pointer to the mode shapes (eigenvectors).
- **ptrLOD**: Pointer to the load vectors.
- **ptrSTF**: Pointer to the reduced stiffness matrix.
- **ptrMAS**: Pointer to the reduced mass matrix.
- **ptrDMP**: Pointer to the reduced damping matrix.
- **ptrCOF**: Pointer to the mode coefficients.
- **ptrDCF**: Pointer to the modal damping coefficients.
- **ptrLPM**: Pointer to the diagonal mass vector.
- **ptrSP1**: Pointer to the spectrum data.
- **ptrIRHSl,h**: Pointer to the imaginary part of the RHS vector.
- **PowerDyn**: PowerDynamics key (currently only set in subout.F).

## DOF Reference Numbers

- **UX**: 1, **UY**: 2, **UZ**: 3, **ROTX**: 4, **ROTY**: 5, **ROTZ**: 6, **AX**: 7, **AY**: 8
- **AZ**: 9, **VX**: 10, **VY**: 11, **VZ**: 12

### Nodal Equivalence Table

- **Back(i)**: i=1,lenbac

### Reduced Set of Degrees of Freedom

- **l(i)**: i=1,nmrow

### Frequencies

- **freq(i)**: i=1,nmode

### Participation Factors

- **pfact(i)**: i=1,nmode

### Mode Shapes

- **modecf(i)**: i=1,nmode

### Modal Damping Coefficients

- **dampmd(i)**: i=1,nmode

### Mode Shapes

- **spectrum**

The MODE file format allows for the storage of complex modes, which are essential for understanding the behavior of structures under dynamic loads.
### Description of the Element Matrices File

This section explains the content of the element matrices file (jobname.emat).

#### 1.6.1. Standard ANSYS File Header

See Section 1.1.2: The Standard Header for ANSYS Binary Files for a description of this set. File number (Item 1) is 2.

#### 1.6.2. EMAT File Format

```plaintext
*comdeck,fdemat
*** copyright(c) 2006 SAS IP, Inc.  All rights reserved.
*** ansys, inc.

********** description of element matrix file **********

*** mpg fdemat.inc < eoelem elostr eofini outelm elfini EmatAssemble sffini
c egprep sfform elstrt slvstr: emat file description
c
character*8 EMATNM
parameter (EMATNM='emat')

LONGINT  ematfpL, ematfp
integer  ematbk, ematut
common /fdemat/ ematfpL, ematbk, ematut
```

#### Table: Nodal Equivalence Table

- **LOD**  dp  1  nmrow  Load vector. This record is present only if extopt=0 or 1.
- **IRHS** dp  1  nmrow  Imaginary Load vector. This record is present only if extopt = 6.
- **LPM** dp  1  nmrow  Lumped mass vector. This record is present only if lumpms=1 and nmatrix=0. It is a vector containing the mass at each node in the system.
- **STF** dp  nmrow nmrow  Reduced stiffness matrix. Each row of the matrix is stored as a record. The matrix is present only if nmatrix > 0. Row order is the same as the DOF order stored at position ptrRDF.
- **MAS** dp  nmrow nmrow  Reduced mass matrix. Each row of the matrix is stored as a record. The matrix is present only if nmatrix > 1. Row order is the same as the DOF order stored at position ptrRDF.
- **DMP** dp  nmrow nmrow  Reduced damping matrix. Each row of the matrix is stored as a record. The matrix is present only if nmatrix > 2. Row order is the same as the DOF order stored at position ptrRDF.

For each spectrum (nspect records):
- **SP1** dp  1  nmode  Mode coeff for this spectra
- **---** dp  1  nmode  Modal damping values
- **---** dp  1  130  svcom: freqtb, etc.
- **---** dp  1  20  misc. spectra data
equivalence (ematfp, ematfpl)

c ************ common variable descriptions ************
co ematfpl      file position on file emat
co ematbk       block number for file emat
co ematut       file unit for file emat

c See fddesc for documentation of how binary files are stored.
c
c ************ file format ************

c recid tells the identifier for this record. Not all records will have
identifiers -- they are only indicated for those records whose
record pointers are stored in the second file header.

c type tells what kind of information is stored in this record:
  i - integer
  dp - double precision
  cmp - complex

c nrec tells how many records of this description are found here

c lrec tells how long the records are (how many items are stored)

<table>
<thead>
<tr>
<th>recid</th>
<th>type</th>
<th>nrec</th>
<th>lrec</th>
<th>contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>---</td>
<td>i</td>
<td>1</td>
<td>100</td>
<td>standard ANSYS file header (see binhed for details of header contents)</td>
</tr>
<tr>
<td>---</td>
<td>i</td>
<td>1</td>
<td>40</td>
<td>.EMAT FILE HEADER</td>
</tr>
</tbody>
</table>

  - fun02, nume, numdof, lenu, lenbac,
  - maxn, 0, 0, nodref, lumpm,
  - kygst, kygm, kycd, kygss, kygaf,
  - kycrf, 0, 0, 0, 0,
  - ptrElmh, ptrFSTh, ptrLSTh, ptrBITh, ptrEHDh,
  - ptrIDXh, numCE, maxLeng, ptrCEl, ptrCEh,
  - ptrDOF, ptrBAC, ptrELMl, ptrFSTl, ptrLSTl,
  - ptrBITl, ptrEHDl, ptrIDXl, ptrendH, ptrendL

each item in header is described below:

- fun02  - unit number (emat file is 2)
- nume  - number of elements
- numdof - number of dofs per node
- lenu  - total DOFs of model
- lenbac - number of nodes
- maxn  - maximum node number
- 0    - position not used
- 0    - position not used
- nodref - actual number of nodes referenced
- lumpm - lumped mass key
  - 0 - default matrix type
  - 1 - lumped
- kygst - global stiffness matrix calculate key
  - 0 - do not calculate
  - 1 - calculate
- kygm  - global mass matrix calculate key
  - 0 - do not calculate
  - 1 - calculate
- kycd  - global damping matrix calculate key
  - 0 - do not calculate
  - 1 - calculate
- kygss - global stress stiffening matrix calculate key
  - 0 - do not calculate
  - 1 - calculate
- kygaf - global applied force vector calculate key
  - 0 - do not calculate
  - 1 - calculate
kygrf - global restoring force vector
   calculate key (Newton-Raphson only)
   = 0 - do not calculate
   = 1 - calculate
   0 - position not used
   0 - position not used
   0 - position not used
   0 - position not used

ptrELMh- High pointer to element equivalence table
ptrFSTh- High pointer to first element at a DOF table
ptrLSTh- High pointer to last element at a DOF table
ptrBITh- High pointer to dof bits
ptrEHDh- High pointer to the start of the element matrices
ptrIDXh- High pointer to element matrices index table

numCE - number of internal CEs
maxLeng- maximum length of any internal CE
ptrCEL - low pointer to internal CE list
ptrCElh - high pointer to internal CE list
ptrDOF - pointer to degrees of freedom per node used in model
ptrBAC - pointer to nodal equivalence table

ptrELMl- Low pointer to element equivalence table
ptrFSTl- Low pointer to first element at a DOF table
ptrLSTl- Low pointer to last element at a DOF table
ptrBITl- Low pointer to dof bits
ptrEHDl- Low pointer to the start of the element matrices
ptrIDXl- Low pointer to element matrices index table

ptrendH- High pointer to end of file
ptrendL- Low pointer to end of file

Note: the analysis type sets the global calculate keys.

--- dp 1 20 Time information

     timval, timinc, frqval, timbeg, timend,
     0.0,  0.0,  0.0,  0.0,  0.0,
     0.0,  0.0,  0.0,  0.0,  0.0,
     0.0,  0.0,  0.0,  0.0,  0.0,

each item is described below:

timval - the current time
timinc - the time increment
frqval - the current frequency (from a harmonic analysis)
timbeg - the start time for the analysis
timend - the end time for the analysis
0.0 - position not used
0.0 - position not used
0.0 - position not used
0.0 - position not used
0.0 - position not used
0.0 - position not used
0.0 - position not used
0.0 - position not used
0.0 - position not used
0.0 - position not used
1.6.2. EMAT File Format

c DOF i 1 numdof Degrees of freedom per node
c DOF reference numbers are:
  UX = 1, UY = 2, UZ = 3, ROTX = 4, ROTY = 5, ROTZ = 6, AX = 7, AY = 8
  AZ = 9, VX = 10, VY = 11, VZ = 12 ****** 13-18 are spares **********
  EMF = 25, CURR = 26 ****** 27-32 are spares ********************
  (curdof(i), i=1, numdof)

c BAC i 1 lenbac Nodal equivalence table. This table equates
  the number used for storage to the actual
  node number
  (Back(i), i=1, lenbac)

c ELM i 1 nume Element equivalence table. The ANSYS program
  stores all element data in the numerical
  order that the SOLUTION processor solves the
  elements. This table equates the order
  number used to the actual element number
  (Order(i), i=1, nume)

c FST i 1 lenu First element at a DOF table. This record
  signifies the first element encountered at a
  particular DOF.
  (First(i), i=1, lenu)

c LST i 1 lenu Last element at a DOF table. This record
  signifies the last element encountered at a
  particular DOF.
  (Last(i), i=1, lenu)

c BIT i 1 lenu Bits set at a DOF table. This record
  has bits for constraints, forces, etc.
  (DofBits(i), i=1, lenu) (added at 10.0)

c IDX i 1 2*nume Element index table. This record specifies
  the file location for the beginning of the
  data for each element.
  (index(i), i=1, nume) Low part of pointer
  (index(i), i=1, nume) High part of pointer

The records at the end of the file store element information and get written
as a set for each element (nume sets of these records will appear on the file
at this point) ptrEHD indicates the beginning of the element data.

If substructure matrices are written to the EMAT file, they are written in a
different format than is shown here. This alternate format is not documented
c at this time, as it is likely to change in the future.

c EHD i 1 10 Element matrix header

  stkey, mkey, dkey, sskey, akey,
  nrkey, ikey, 0, 0, nmrow

each item in header is described below:

c stkey - stiffness matrix key
  0 - matrix not present
  1 - matrix present

c mkey - mass matrix key
  0 - matrix not present
  1 - matrix present

c dkey - damping matrix key
  0 - matrix not present
  1 - matrix present

c sskey - stress stiffening matrix key
  0 - matrix not present
  1 - matrix present
Chapter 1: Format of Binary Data Files

1.7. Description of the Substructure Matrices File

This section explains the contents of the substructure matrices file (jobname.sub).
1.7.1. Standard ANSYS File Header

See Section 1.1.2: The Standard Header for ANSYS Binary Files for a description of this set. File number (Item 1) is 8.

1.7.2. SUB File Format

`*comdeck, fsub
   c *** copyright (c) 2006 SAS IP, Inc. All rights reserved.
   c *** ansys, inc
   c
   ************ description of substructure matrix file ************
   character*8 SUBNM
   parameter (SUBNM='sub    ')
   LONGINT subfpL, lenSubL
   integer subbk, subut
   common /fdsub/ subfpL, lenSubL, subbk, subut
   c write: matout
   c read:
   c
   ************ common variable descriptions ************
   co subfpL      file position on file sub
   co subbk       block number for file sub
   co subut       file unit for file sub
   co lenSubL     length of sub file (saved for slvdta.F)
   c
   See fddesc for documentation of how binary files are stored.
   c
   ************ file format ************
   c
   recid tells the identifier for this record. Not all records will have
   c identifiers -- they are only indicated for those records whose
   c record pointers are stored in the second file header.
   c
   type tells what kind of information is stored in this record:
   c      i - integer
   c      dp - double precision
   c      cmp - complex
   c
   nrec tells how many records of this description are found here
   c
   lrec tells how long the records are (how many items are stored)
   c
   recid    type    nrec    lrec     contents
   c
   ---      i       1      100      standard ANSYS file header (see binhed
                                 for details of header contents)
   c
   HED      i       1       60      .SUB FILE HEADER (FULL MATRICES)
   c
   8, nmrow, nmatrix, nedge, numdof,
   c
   maxn, wffmax, lenbac, nnod, kunsym,
   c
   kstf, kmass, kdamp, kss, nvect,
   c
   nWorkL, lenU1, sesort, lenlst, ptrLodL,
   c
   ntrans, ptrMtx, ptrXFM, ptrHED, name1,
   c
   name2, 0, 0, name3, name4,
   c
   ptrDOF, ptrGST, ptrBAC, ptrTIT, ptrNOD,
   c
   ptrXYZ, ptrEDG, ptrGDF, thsubs, ptrPOS,
   c
   ptrORG, stfmax, ptrLodH, nmodes, keydim,
   c
   cmuMethod, name5, name6, name7, name8,
   c
   nvnodes, ptrCTXM, nWorkh, ptrCG, 0,
   c
   0, 0, 0, 0, 0
   c
   HED      i       1       60      .SUB FILE HEADER (SPARSE MATRICES)
   c
   9, nEqn, nmatrix, , numdof,
   c
   maxn, , lenbac, nnod, kunsym,
   c
   kstf, kmass, kdamp, , nvect,`
Chapter 1: Format of Binary Data Files

each item in header is described below:

fun08 - unit number (full sub file is 8)
  (sparse substructure file is 9)

nmrow - number of rows in matrices (also
  number of dofs in substructure)

nmattx - number of matrices on file

nedge - number of edges for outline

numdof - number of dofs per node

maxn - maximum node number of complete
  model presently in database

wfmax - maximum wavefront of substruct.
  during generation pass

lenbac - number of nodes defining
  substructure during the
  generation pass

nnod - number of unique nodes in the
  substructure having DOFs, and
  which define this substructure
  during the use pass. Also, the
  number of nodes having master
  DOFs.

kunsym - unsymmetric matrix key
  = 0 - symmetric
  = 1 - unsymmetric

kstf - stiffness matrix present key
  = 0 - matrix is not on file
  = 1 - matrix is on file

kmass - mass matrix present key
  = 0 - matrix is not on file
  = 1 - matrix is on file
  =-1 - Lumped mass vector (Sparse only)

kdamp - damping matrix present key
  = 0 - matrix is not on file
  = 1 - matrix is on file

kss - stress stiffening matrix present
  = 0 - matrix is not on file
  = 1 - matrix is on file

nvect - number of load vectors
  (at least 1 is required)

nWorkL,H - BCS workspace length (only for
  bacsbst)

nTermL,H - Number of terms in sparse
  matrix

lenU1 - length of intermediate transformation
  vector

sesort - DOF set sort key
  = 0 - numbers are not sorted
  = 1 - numbers are sorted in
  ascending order

lenlst - maximum length of DOF set for
  this substructure (maxn*numdof)

ptrLod - pointer to the start of the load
  vectors (see also ptrLodh)

ntrans - transformed key
  = 0 - substructure has not been
  transformed
  > 0 - substructure copied from another substructure,
  via either SESSYM or SETRAN

ptrMtxL,H - pointer to the start of the
  substructure matrices (iDiagL for
  sparse matrices)
ptrXFM - pointer to the substructure transformations
ptrHED - pointer to the SUB file header
name1 - first four characters of the substructure file name, in integer form
name2 - second four characters of the substructure file name, in integer form
name3 - third four characters of the substructure file name, in integer form
name4 - fourth four characters of the substructure file name, in integer form
ptrDOF - pointer to the DOF/node list
ptrDST - pointer to the local DOF set of the substructure
ptrTIT - pointer to the title
ptrNOD - pointer to the unique nodes defining the substructure
ptrXYZ - pointer to the coordinates of the unique nodes
ptrEDG - pointer to the substructure edges
ptrGDF - pointer to the global DOF set
ptrCG - pointer to the element mass information
thsubs - thermal key
    = 0 - structural
    = 1 - thermal
ptrPOS - pointer to the sorted substructure DOF set to the original
ptrORG - pointer to the DOF set of the model during the generation pass
stfmax - maximum diagonal stiffness term (packed into an integer)
ptrLodh - High 32 bits of 64 bit pointer
nmodes - number of modes used to generate CMS s.e.
keydim - dimensionality key
    = 1 - axisymmetric
    = 2 - 2-D
    = 3 - 3-D
cmsMethod - component mode synthesis method
name5 - fifth four characters of the substructure file name, in integer form
name6 - sixth four characters of the substructure file name, in integer form
name7 - seventh four characters of the substructure file name, in integer form
name8 - eighth four characters of the substructure file name, in integer form
nvnodes - number of virtual nodes that contain the modal coordinates
ptrCTXM - coordinate transformation
ptrColL,H - pointer to the iCol sparse matrix array
ptrCofL,H - pointer to the of the sparse matrix Sk(1:nTerm), Sm(1:nTermL), Sc(1:nTermL), Ss(1:nTermL) Each matrix is a single large record

note: name1/2/3/4/5/6/7/8 are the inexc4 representation of the 32 character filename.
name1/2/5/6/7/8 will be "0" for pre rev 5.2 files - cwa
for pre rev 5.2 files - cwa
Note: If ntrans > 0, records from position ptrDOF to ptrGDF will be identical to the data for the copied substructure.

XFM dp 1 125  Substructure transformations (5*25 double precisions). This record has meaning only if ntrans > 0. You can define up to five levels of transformations, with 25 variables in each level. Up to the first seven variables are used as follows:

If the substructure was transferred (via the SETRAN command):
1st variable - 1.0
2nd variable - nodal increment
3rd variable - reference number of coordinate system where substructure will be transferred
4th variable - reference number of coordinate system where substructure is presently defined
5th variable - x coordinate increment
6th variable - y coordinate increment
7th variable - z coordinate increment

If the substructure used symmetry (via the SESYMM command):
1st variable - 2.0
2nd variable - nodal increment
3rd variable - number of coordinate component to be used in operation
= 1 - x coordinate
= 2 - y coordinate
= 3 - z coordinate
4th variable - reference number of coordinate system to be used for symmetry operation

CTXM dp 1 250  Substructure transformations

DOF i 1 numdof Degrees of freedom per node (Global)
(curdof(i),i=1,numdof)
DOF reference numbers are:
UX  = 1, UY  = 2, UZ  = 3, ROTX= 4, ROTY= 5, ROTZ= 6, AX  = 7, AY  = 8
AZ  = 9, VX  =10, VY  =11, VZ  =12  ***** 13-18 are spares ******
EMF =25, CURR=26  ********* 27-32 are spares *************************

DST i 1 nmrow This record contains degrees of freedom for this substructure of the unique nodes, as used with this substructure, in ascending order. This index is calculated as (N-1)*numdof+DOF, where N is the node number and DOF is the DOF reference number given above
(lsort(i),i=1,nmrow)

POS i 1 nmrow This record stores the positions of the local DOF set in relation to the generated DOF set. (lposit(i),i=1,nmrow)

ORG i 1 nmrow DOF set of the model as defined during the generation pass. This index is calculated as (N-1)*NUMDOF+DOF, where N is the position number of the node in the nodal equivalence table and DOF is the DOF reference number given above
(lorig(i),i=1,nmrow)

BAC i 1 lenbac This group describes nodes that defined the substructure during the generation pass of the analysis. Nodal data is stored in arrays
equal to the number of used or referenced nodes. This table equates the number used for storage to the actual node number.

(Back(i), i=1, lenbac)

Substructure title (converted to integers - see inexc4)

This record describes unique nodes defining the substructure for the use pass of the analysis. These are also the nodes having master degrees of freedom.

(node(i), i=1, nnod)

This record describes the coordinates of a unique node, in the order X, Y, Z, THXY, THYZ, and THZX. Nodal order corresponds to that of the node list given above

(xyzang(j,i), j=1, 6)

This record contains beginning and ending locations (X1, Y1, Z1, X2, Y2, Z2 coordinates) of a straight line comprising an edge of the substructure.

This record describes global degrees of freedom of the unique nodes in ascending order, as used during the analysis use pass. This index is calculated as (N-1)*32+DOF, where N is the node number and DOF is the DOF reference number given above

(l(i), i=1, nmrow) (sorted)

total mass, CGx, CGy, CGz, 6 moments of inertia

The substructure matrices are written at this position in the file. One row of each matrix is written to the file at a time. i.e. the first row of each matrix is written, then the second row of each matrix, etc. this pattern continues until all nmrow rows of each matrix have been written to the file.

Row of the stiffness matrix, if nmatrx > 0.

(ak(i,j), i=1, nmrow)

Row of the mass matrix, if nmatrx > 1.

(am(i,j), i=1, nmrow)

Row of the damping matrix, if nmatrx > 2.

(ac(i,j), i=1, nmrow)

Row of the stress stiffening matrix, if nmatrx > 3.

(gs(i,j), i=1, nmrow)

This record contains the load vectors.

(f(i), i=1, nmrow)

1.8. Description of the Component Mode Synthesis Matrices (CMS) File

This section explains the contents of the CMS matrices file (jobname.cms).

1.8.1. Standard ANSYS File Header

See Section 1.1.2: The Standard Header for ANSYS Binary Files for a description of this set. File number (Item 1) is 8.

1.8.2. CMS File Format

*comdeck, fdcms

c --- description of cms(component modal synthesis) transformation file
Chapter 1: Format of Binary Data Files

character*8 CMSNM
parameter (CMSNM='cms ')
LONGINT cmsfpL, cmsfp
integer cmsbk, cmsut

common /fdcms/ cmsfpL, cmsbk, cmsut
equivalence (cmsfp,cmsfpL)

c --- common variable description -----------------------------------------------
co cmsfp file position on file mode
co cmsbk block number for file mode
co cmsut file unit for file mode

c --- See fddesc for documentation of how binary files are stored. ---------------

c --- file format
c recid tells the identifier for this record. Not all records will have
c identifiers -- they are only indicated for those records whose
c record pointers are stored in the second file header.

c --- type tells what kind of information is stored in this record:
c i - integer
c dp - double precision
c cmp - complex

c --- nrec tells how many records of this description are found here

c --- lrec tells how long the records are (how many items are stored)

c recid type nrec lrec contents

c --- i 1 100 standard ANSYS file header (see binhed for
details of header contents)

c --- i 1 40 .CMS FILE HEADER

c fun45, neqn , nirfm, nnorm, ncstm,
c 0, 0, 0, 0, 0, 0,
c 0, 0, 0, 0, 0, 0,
c 0, 0, 0, 0, 0, 0,
c 0, 0, 0, 0, 0, 0,
c ptrIRFS,ptrNORS,ptrCSTS, 0,ptrIRFL,
c ptrNORL,ptrCSTL, 0, 0, 0,

each item in header is described below:
cc fun45 - unit number
c neqn - number of equation in BCS
c nirfm - number of inertia relief modes
c nnorm - number of normal modes
c ncstm - number of constraint modes
  > 0 available in file
  < 0 NA in file
c ptrIRFS,ptrIRFL - pointer to inertia relief modes
c ptrNORS,ptrNORL - pointer to normal modes
c ptrCSTS,ptrCSTL - pointer to constraint modes
c 0 - position not used

c --- i 1 neqn BCS to ANS mapping (1BCStoANS(i), i= 1,neqn)
c NOR dp nnorm neqn Normal Modes
c IRF dp nirfm neqn Inertia Relief Modes
c CST dp ncstm neqn Constraint Modes
1.9. Description of the Triangularized Stiffness File

This section explains the contents of the triangularized stiffness file (jobname.tri).

1.9.1. Standard ANSYS File Header

See *Section 1.1.2: The Standard Header for ANSYS Binary Files* for a description of this set. File number (Item 1) is 11.

1.9.2. TRI File Format

```plaintext
*comdeck,ftri
** Copyright(c) 2006 SAS IP, Inc. All rights reserved.
** ansys, inc
**
** **********  description of triangularized stiffness file **********
** mpg fdttri.inc < stff10 slvstr romstr modstr: tri file description
character*8  trinm, TriName
parameter (trinm='tri   ',TriName='Tri Bufr')

LONGINT  trifp
integer  tribk,  triut
common /fdtri/   trifp,  tribk,  triut

open:     slvstr                     subspc
write:    eqsol,eqclos,lfout,cfout   subtri
read:     stff10,bacfil              subfwd,subbac
close:    bacfil,svkan2              subspc

** ********** common variable descriptions **********
cotriifp file position on file tri (LONGINT)
cotribk block number for file tri
ocotriut file unit for file tri

c See fddesc for documentation of how binary files are stored.
c
The TRI file is generated for static and reduced[modal,harmonic,transient,
c substructure] analyses
c
** ********** file format **********
c recid tells the identifier for this record. Not all records will have
c identifiers -- they are only indicated for those records whose
c record pointers are stored in the second file header.
c
type tells what kind of information is stored in this record:
i - integer
dp - double precision
cmp - complex
c
nrec tells how many records of this description are found here
c
lrec tells how long the records are (how many items are stored)
c
recid type nrec lrec contents
--- i 1 100 standard ANSYS file header (see binhed for
details of header contents)
--- i 1 20 .TRI FILE HEADER
    fun11, nontp, nmast, i, kan,
    wmax, lenbac, numdof, ptrMS1, ptrrend,
    lumpm, keyuns, ptrMS2, ptrEN1,
    ptrEN2, 0, 0, ptrTRI, 0
```

Programmer's Manual for ANSYS . ANSYS Release 11.0 . 002328 . © SAS IP , Inc. All rights reserved.
Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.
Chapter 1: Format of Binary Data Files

each item in header is described below:

fun11  - unit number (tri file is 11)
nontp  - number of equations on file
nmast  - number of master dofs
  1   - position not used, always = 1
kan    - analysis type
wfmax  - maximum wavefront
lenbac - number of nodes
numdof - number of degrees of freedom (DOF) per node
ptrMST - 32 bit pointer to the master dof list, only here for backward compatibility. Do not use if ptrMS1 or ptrMS2 are non-zero
ptrend - 32 bit pointer to the end of file only here for backward compatibility. Do not use if ptrEN1 or ptrEN2 are non-zero
lumpm - lumped mass key
  = 0 - default matrix type
  = 1 - lumped
keyuns - unsymmetric key
  = 0 - the matrix is not unsymmetric
  = 1 - the matrix is unsymmetric
ptrMS1,
ptrMS2 - These two values are two halves of a 64 bit pointer that points to the master dof list
ptrEN1,
ptrEN2 - These two values are two halves of a 64 bit pointer that points to the end of file
  0   - position not used
  0   - position not used
ptrTRI - pointer to the beginning of the triangularized matrix data
  0   - position not used

--- i 1 numdof Degrees of freedom per node
DOF reference numbers are:
UX  = 1, UY  = 2, UZ  = 3, ROTX= 4, ROTY= 5, ROTZ= 6, AX  = 7, AY  = 8
AZ  = 9, VX  =10, VY  =11, VZ  =12 ***** 13-18 are spares **********
 ********** PRES=19, TEMP=20, VOLT=21, MAG =22, ENKE=23, ENDS=24
EMF =25, CURR=26  27-32 are spares  ********************
(curdof(i),i=1,numdof)

--- i 1 lenbac Nodal equivalence table. This table equates the number used for storage to the actual node number
(Back(i),i=1,lenbac)

TRI At this point in the file, the triangularized matrix information is stored. The info is written row by row, and there are two different storage options for writing a row. If the row being written does not have a constraint equation associated with it, then two records are written to describe the row. If the row being written has a constraint equation associated with it, then five records are written to describe the row. Both formats are shown below. These groupings of two or five records per row will be written a total of nontp times (to include all rows)

The next two descriptions show the format for a row that does not have a constraint equation associated with it:

--- dp/cmp 1 varies A row of the triangularized matrix.

If keyuns=0, this record will contain the non-diagonal terms of this column, the diagonal term itself, the normalized F term, followed by the reciprocal of the row
If keyuns=1, this record will contain the non-diagonal terms of this column, the diagonal term itself, the normalized F term, the reciprocal of the row pivot, followed by the non-diagonal terms of this row.

The length of this record will vary (actual length is returned from routine BINRD8). If kan=3, this record contains complex information, otherwise it contains double precision information.

If keyuns=0, this record will contain the non-diagonal terms of this column, the diagonal term itself, the normalized F term, followed by the reciprocal of the row pivot.

The length of this record will vary (actual length is returned from routine BINRD8).

If keyuns=1, this record will contain the non-diagonal terms of this column, the diagonal term itself, the normalized F term, the reciprocal of the row pivot, followed by the non-diagonal terms of this row.

The length of this record will vary (actual length is returned from routine BINRD8). If kan=3, this record contains complex information, otherwise it contains double precision information.

Triangular matrix row indices. The first item signifies what term in the row belongs to the pivot. The second term signifies what DOF is being eliminated, and the remaining items signify the new DOFs being introduced (if any). The length of this record will vary (actual length is returned from routine BINRD8).

The next five descriptions show the format for a row that has a constraint equation associated with it.

A flag record, indicating that constraint equations are being stored, and the storage is as shown here. Both values are TINY.

Coefficients of the constraint equation. The length of this record will vary (actual length is returned from routine BINRD8).

A row of the triangularized matrix.

Triangular matrix row indices. The first item signifies what term in the row belongs to the pivot. The second term signifies what DOF is being eliminated, and the remaining items signify the new DOFs being introduced (if any). The length of this record will vary (actual length is returned from routine BINRD8).
1.10. Description of the Full Stiffness-Mass File

This section explains the contents of the full file (jobname.full).

1.10.1. Standard ANSYS File Header

See Section 1.1.2: The Standard Header for ANSYS Binary Files for a description of this set. File number (Item 1) is 4.

1.10.2. FULL File Format

*comdeck,fdfull
** *** copyright(c) 2006 SAS IP, Inc. All rights reserved.
** *** ansys, inc.

** ********** description of full stiffness-mass file **********
** *** mpg fdfull.inc < stff10 slvstr: full file description

character*8  FULLNM
parameter (FULLNM='full ')

** *** NOTE: if this variable is changed in the future it should be
** *** updated in spdefines.h also for symbolic assembly (jrb)
integer    FULLHDLEN
parameter (FULLHDLEN=60)

LONGINT    fullfpL, fullfp
integer    fullbk, fullut, wrldstep, wrsbstep, wrEqiter
common /fdfull/ fullfpL, fullfp, fullbk, fullut, wrldstep, wrsbstep, wrEqiter
x           wrldstep,wrsbstep,wrEqiter
 equivalence (fullfp,fullfpL)

** ********** common variable descriptions **********
co fullfpL  file position on file full
co fullbk   block number for file full
co fullut   file unit for file full

** ********** file format (except for extopt=3,4) **********
co See fddesc for documentation of how binary files are stored.

** ********** file format **********

c recid tells the identifier for this record. Not all records will have
 identifiers -- they are only indicated for those records whose
record pointers are stored in the second file header.

type tells what kind of information is stored in this record:
i - integer
dp - double precision
cmp - complex

nrec tells how many records of this description are found here

lrec tells how long the records are (how many items are stored)

---
recid  type  nrec  lrec  contents
---
  i  1  100  standard ANSYS file header (see binhed for details of header contents)
  i  1  60  .FULL FILE HEADER

NOTE: If fun04 > 0, then the file was created with frontal assembly

If fun04 < 0, then the file was created with symbolic assembly; see below for its format

-------------------------- frontal assembled file --------------------------

each item in header is described below:

fun04 - unit number (full file is 4)
neqn - number of equations on file
nmrow - number of rows in matrices
nmatrix - number of matrices on file
kan - analysis type
wfmax - maximum wavefront
lenbac - number of nodes
numdof - number of dofs per node
jcgtrmL, jcgtrmH - number of coefficients
lumpm - lumped mass key
  0 - default matrix type
  1 - lumped
jcgeqn - number of jcgeqn equations
jcgtrm - pre-8.1 this is the number of coefficients in sparse jcg
  storage (otherwise this value must be 0 and jcgtrmL, jcgtrmH must be used)
keyuns - unsymmetric key
  0 - no unsymmetric matrices on file
  1 - there is at least one unsymmetric matrix on file
extopt - mode extraction method
  0 - reduced
  1 - lumped
  3 - unsymmetric Lanczos
  4 - damped Lanczos
  6 - block Lanczos
keyse - superelement key; set if at least one superelement
sclstf - scale factor for matrices
nxrows - the maximum rank for this solution
ptrIDXl - pointer to the matrix row indices.
ptrIDXh - high part of row index pointer
ncefull - Number of constraint equations on
the full file
ncetrm - Total number of terms in the
constraint equations
ptrENDl - Low part of 64 bit end of file ptr
ptrENDh - High part of 64 bit end of file ptr
0 - position not used

--- i 1 numdof Degrees of freedom per node
Dof reference numbers are:
UX = 1, UY = 2, UZ = 3, ROTX= 4, ROty = 5, ROTZ= 6, AX = 7, AY = 8
AZ = 9, VX =10, VY =11, VZ =12 ****** 13-18 are spares ************
EMF =25, CURR=26 ****** 27-32 are spares ***************
(curfld(i),i=1,numdof)

--- i 1 lenbac Nodal equivalence table. This table equates
the number used for storage to the actual
node number
(Back(i),i=1,lenbac)

NOTE: The next five records are repeated as a group neqn times.
When the matrices get written, one row of each matrix is written to the file
at a time. i.e. the first row of each matrix is written, then the second row
each of each matrix, etc. this pattern continues until all the rows of each
matrix have been written to the file. If ka=3, the matrix rows will be
complex valued, otherwise they will be double precision values.

IDX i 1 varies Matrix row indices. The first
item signifies what term in the row belongs
to the pivot. The second term signifies what
Dof is being eliminated, and the remaining
items signify the new Dofs being introduced
(if any). The length of this record will vary (actual length is returned from routine
BINRD8)
(lll(i),i=1,m)

--- i 1 varies A second level of indexing for the
matrix. Indicates positions and
values of terms to be reduced. The length of
this record will vary (actual length is
returned from routine BINRD8)
(index(i),i=1,n) for compressed rows

--- dp/cmp 1 varies Stiffness matrix.

If keyuns=0, this record will contain the
non-diagonal terms of this column, the
diagonal term itself, followed by the
normalized F term.

If keyuns=1, this record will contain the
non-diagonal terms of this column, the
diagonal term itself, the non-diagonal terms
of this row, followed by the normalized F
term.

If lumpm = 1, then the mass for this node is
located after the F term. The length of
this record will vary (actual length is
returned from routine BINRD8)
(krow(i),i=1,n),vload,(mass) (symmetric)
(n-1 column) diag (n-1 row) load (dmass)
(unsymmetric)

--- dp/cmp 1 varies Mass matrix. This record exists only if
nmatrix > 1.
If keyuns=0, this record will contain the
non-diagonal terms of this column, the
diagonal term itself, followed by the
normalized F term.

If keyuns=1, this record will contain the
non-diagonal terms of this column, the
diagonal term itself, followed by the
non-diagonal terms of this row.

The length of this record will vary (actual
length is returned from routine BINRD8)
(mrow(i), i=1,n) (symmetric)
(n-1 column) diag (n-1 row) (unsymmetric)

If lumpms=1, this record contains one double
array with diag values

--- dp/cmp 1 varies Damping matrix. This record exists only if
--- nmarrx > 2.

--- i 1 60 .FULL FILE HEADER

--- i 1 60 .FULL FILE HEADER

--- fun04, negn, nmrow, nmarrx, kan,
--- wfname, lenbac, numdof, ntermKl, ntermKh,
--- lumpm, nmrow, ntermK, keyuns, extopt,
--- keyse, sclstf, nxrows, ptrSTFl, ptrSTFh,
--- ncefull, ntermMh, ptrENDl, ptrENDh, ptrIRHSL,
--- ptrIRHSh, ptrMASl, ptrMASh, ptrDMPFl, ptrDMPFh,
--- ptrCEl, ptrCEh, nNodes, ntermML, ntermDl,
--- ntermDOFl, ptrDOFl, ptrDOFh, ntermRhsL, ntermRhsH,
--- ngMaxNZ,ptrNGPHl,ptrNGPHh,minKdiag,maxKdiag,
--- minMdiag,maxMdiag,minDdiag,maxDdiag, ngTermi,
--- ngTermh,ngTermCl,ngTermCh, 0, 0,
--- 0, 0, 0, 0, 0

--- each item in header is described below:

fun04  - negative of the unit number (-4)
negn  - number of equations on file
nmrow  - number of active DOF (negn-BC)
nmarrx - number of matrices on file
kan    - analysis type
wfname - maximum row size
lenbac - number of nodes
numdof - number of dofs per node
ntermKl,ntermKh - number of terms in Stiffness
matrix
lumpm  - lumped mass key
--- 0 - default matrix type
--- 1 - lumped
ntermK - pre-8.1 this is the number of terms
in Stiffness matrix (otherwise this
value must be 0 and ntermKl,ntermKh
must be used)
Chapter 1: Format of Binary Data Files

- `keyuns`: unsymmetric key
  - 0 - no unsymmetric matrices on file
  - 1 - there is at least one unsymmetric matrix on file

- `extopt`: mode extraction method
  - 0 - reduced
  - 1 - lumped
  - 3 - unsymmetric Lanczos
  - 4 - damped Lanczos
  - 6 - block Lanczos
  - 7 - QRdamped
  - 8 - AMLS (not implemented)
  - 9 - PCG Lanczos

- `keyse`: superelement key; set if at least one superelement

- `sclstf`: maximum absolute stiffness matrix term

- `nxrows`: the maximum rank for this solution

- `ncefull`: number of CE+CP equations

- `ptrENDl`, `ptrENDh`: low and high part of 64 bit end of file pointers

- `ptrIRHSl`, `ptrIRSHsl`: pointer to imaginary RHS (F)

- `ptrMASl`, `ptrMASh`: pointer to Mass matrix

- `ptrDMPl`, `ptrDMPh`: pointer to Damping matrix

- `ptrCEl`, `ptrCEh`: pointer to Gt and g matrices

- `nNodes`: number of internal Nodes

- `ntermMl`, `ntermMh`: number of terms in Mass matrix

- `ntermDl`, `ntermDh`: number of terms in Damping matrix

- `ptrDOFl`, `ptrDOFs`: pointer to DOF info

- `ptrRHSls`, `ptrRSHsh`: pointer to RHS (F)

- `ngMaxNZ`: maximum number of nodes per nodal block in nodal graph structure

- `ptrNGPFl`, `ptrNGPFh`: pointer to vectors needed for nodal graph structure

- `minKdiag`, `maxKdiag`: minimum and maximum absolute stiffness matrix diagonal term

- `minMdiag`, `maxMdiag`: minimum and maximum absolute mass matrix diagonal term

- `minDdiag`, `maxDdiag`: minimum and maximum absolute damping matrix diagonal term

- `ngTerml`, `ngTermCh`: total number of nonzeros in nodal graph (expanded graph based value)

- `ngTermCl`, `ngTermCh`: total number of nonzeros in nodal graph (compressed graph based value)

- `numdof`: Degrees of freedom per node

- `numdof`: DOF reference numbers are:
  - UX = 1, UY = 2, UZ = 3, ROTX= 4, ROTY= 5, ROTZ= 6, AX = 7, AY = 8
  - AZ = 9, VX =10, VY =11, VZ =12 ******** 13-18 are spares *********
  - PRES=19, TEMP=20, VOLT=21, MAG =22, ENKE=23, ENDS=24
  - EMF =25, CURR=26 ********* 27-32 are spares ****************

- `lenbac`: Nodal equivalence table. This table equates the number used for storage to the actual node number

- `STF`: Matrix row indices. The last item corresponds to the diagonal. The length of this record will vary (actual length is returned from routine BNRDB8)
**Load Vector**

- **RHS**: dp/cmp 1 neqn Load vector terms.
- **Imaginary part of Load Vector**
  - **IRHS**: dp 1 neqn Imaginary load vector terms.

**DOF information**

- **DOF**: i 1 nNodes Nodal extent vector. Number of DOFs at each node.
- **DOF vector**: i 1 neqn DOF vector. If negative, this DOF constrained.
- **DOFs with imposed values**: i 1 neqn DOFs with imposed values.

**Mass Matrix.**

- If lumpm = 0:
  - The next two records are repeated as a group neqn times.
- **MAS**: i 1 varies Matrix row indices. The last item corresponds to the diagonal. The length of this record will vary (actual length is returned from routine BINRD8).

**Damping Matrix.**

- The next two records are repeated as a group neqn times.
- **DMP**: i 1 varies Matrix row indices. The last item corresponds to the diagonal. The length of this record will vary (actual length is returned from routine BINRD8).

**G matrix if ncefull > 0.**

- **CE**: i 1 ncefull List of slave DOFs
- **g vector (constant terms)**: dp 1 ncefull Imaginary g vector (constant terms). This vector only exists for harmonic analyses. (Antype = 3).
- **Column indices**: i 1 varies Column indices
- **Column values**: dp 1 varies Column values
c Nodal graph vectors

c NGPH  i   1  nNodes  number of nonzeros for each node

c Repeat for each node

c   i   1  varies  Index vector.

c Meaning of K11, K12, and G matrices:

c   Given

c   [K]{x} = {F}

c   subject to the constraints

c   {x1} = [G]{x2} + {g}

c   where {x1} are the slave DOFs, {x2} the master DOFs

c This results in

c   [K*]{x2} = {F*}

c where

c   [K*] = [G]'[K11][G] + [G]'[K12] + [K21][G] + [K22]

c   {F*} = [G]'{f1} + {f2} - [G]'[K11]{g} - [K21]{g}

complex version of {F*} decomposed into, we assume G' is always real

and g could be complex denoted as g' == (g,gx) :

c   G' K11' g' = G' (K11,M11)*(g,gx)

c     = G' [K11*g - M11*gx, M11*g + K11*gx]

   K21' *g' = (K21,M21)*(g,gx)

c     = (K21*g- M21*gx, K21*gx + M21*g)
Chapter 2: Accessing Binary Data Files

2.1. Accessing ANSYS Binary Files

The following section explains the routines you need to read, write, or modify an ANSYS binary file. This collection of routines (called BINLIB) resides on your ANSYS distribution media. The BINLIB library is in the dynamic link library \Program Files\Ansys Inc\V110\ANSYS\custom\misc\<platform>\binlib.dll (on Windows systems (where <platform> is a directory that uniquely identifies the hardware platform version)) or the shared library /ansys_inc/v110/ansys/customize/misc/<platform>/libbin.so on UNIX systems (libbin.so1 on HP systems).

Your distribution media also includes sample FORTRAN source files which employ the BINLIB library:

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bintrd.F</td>
<td>The bintrd subroutine reads and prints the contents of an ANSYS binary file</td>
</tr>
<tr>
<td>bintwr.F</td>
<td>The bintwr subroutine copies an ANSYS binary file to a new file</td>
</tr>
<tr>
<td>bintst.F</td>
<td>The bintst program calls the bintwr and bintrd subroutines as an example of how to use the binlib library to print the contents of a file, copy the original file to a new file, and then print the contents of the new file. Routine bintst has no inputs or outputs. It requires use of the bintcm common. (For more information, see the descriptions of the bintrd and bintwr routines later in this chapter.)</td>
</tr>
</tbody>
</table>

These files reside in the subdirectory \Program Files\Ansys Inc\V110\ANSYS\custom\misc\<platform> (on Windows systems) or /ansys_inc/v110/ansys/customize/misc (on UNIX systems). To compile and link the BINTST program, a makefile procedure has been provided in the \Program Files\Ansys Inc\V110\ANSYS\custom\misc\<platform> subdirectory on Windows or run the bintst.link procedure in the /ansys_inc/v110/ansys/customize/misc subdirectory on UNIX.

2.1.1. Access Routines to Results and Substructure Files

Demonstration programs for reading and writing ANSYS results and substructure files are included on the installation media:

- ResRdDemo
- ResWrDemo
- rdsubs
- wrtsub
- rdfull

On Windows systems:

The FORTRAN source for these programs is located in \Program Files\Ansys Inc\V110\ANSYS\custom\misc and the files are named ResRdDemo.F, ResWrDemo.F, rdsubs.F, wrtsub.F, and rdfull.F.

To link these demonstration programs, use the \Program Files\Ansys Inc\V110\ANSYS\custom\misc\<platform>\rdrwrt.bat procedure file and specify the program that you want to build on the command line. Valid command line options are ResRdDemo, ResWrDemo, rdsubs, wrtsub, rdfull, and userprog. For example, to build the program to read a results file, type:

\Program Files\Ansys Inc\V110\ANSYS\custom\misc\<platform>\rdrwrt ResRdDemo
Chapter 2: Accessing Binary Data Files

Appropriate files will then be copied from \Program Files\Ansys Inc\V110\ANSYS\custom\misc\<platform> to your working directory, compiled, and linked. The resulting executable will also be placed in your current working directory.

Use the userprog command line option when writing your own customized program, naming the routine userprog.F. The resulting executable will be named userprog.exe. When userprog is used, no files are copied from \Program Files\Ansys Inc\V110\ANSYS\custom\misc\<platform> to your working directory.

These files will be loaded onto your system only if you performed a custom installation and chose to install the customization tools.

On UNIX systems:

The FORTRAN source for these programs is located in /ansys_inc/v110/ansys/custom/misc and the files are named ResRdDemo.F, ResWrDemo.F, rdsubs.F, wrtsub.F, and rdfull.F.

To link these demonstration programs, use the /ansys_inc/v110/ansys/customize/misc/rdrwrt.link procedure file and specify the program that you want to build on the command line. Valid command line options are ResRdDemo, ResWrDemo, rdsubs, wrtsub, rdfull, and userprog.

For example, to build the program to read a results file, type:

```
/ansys_inc/v110/ansys/customize/misc/rdrwrt.link ResRdDemo
```

Appropriate files will then be copied from /ansys_inc/v110/ansys/customize/misc to your working directory, compiled, and linked. The resulting executable will also be placed in your current working directory. Procedure files are available in the /ansys_inc/v110/ansys/bin directory to run these programs, once linked. The procedure files are named ResRdDemo110, ResWrDemo110, rdsubs110, wrtsub110, and rdfull110.

Use the userprog command line option when writing your own customized program, naming the routine userprog.F. The resulting executable will be named userprog.e110. When userprog is used, no files are copied from /ansys_inc/v110/ansys/customize/misc to your working directory. The procedure file is named userprog110.

These files will be loaded onto your system only if you performed a custom installation and chose to install the customization tools.

2.1.2. Characteristics of ANSYS Binary Files

Before accessing ANSYS binary files, you need to know certain file characteristics:

1. An ANSYS binary file is a direct access, unformatted file. You read or write a record by specifying (as a number) what location to read or write.
2. Before the ANSYS program actually writes data to a file on a disk, it uses buffers to store data in memory until those buffers become full. A block number designates these buffers. Most access routines use this block number.
3. By default, ANSYS files are external files. The standardized "external" format the files use enables you to transport them across different computer systems.
4. In addition to file names, ANSYS uses file numbers to identify the files. File handles and other information are associated with the file numbers.
5. Some binary files contain data values that point to the start of certain data (for example, the start of the data steps index table record). Both the ANSYS program and external binary files access routines use these pointers to locate data on the various binary files.

6. All data is written out as 32-bit integers. Double-precision data and pointers, therefore, take up two integer words. To create a 64-bit pointer from the two 32-bit integers, use the function largeIntGet.

2.1.3. Viewing Binary File Contents

To view the contents of certain ANSYS binary files, you issue the command /AUX2 or choose menu path Utility Menu>File>List>Binary Files or Utility Menu>List>File>Binary Files. (You can do so only at the Begin level.) The ANSYS program then enters its binary file dumping processor, AUX2, and dumps the binary file record by record.

AUX2 does not use the data pointers discussed in item 5 above. It uses record numbers to locate the binary file data to dump. Although the information that AUX2 provides includes the pointer, using the pointer alone will not get you that information. To get it, you must correlate the pointer and the record number by trial and error.

2.1.4. Abbreviations

The input and output for the routines discussed in this chapter are described with the following abbreviations:

- **Type** of variable is one of the following:
  - int - integer
  - dp - double-precision
  - log - logical (true or false)
  - char - character

- **Size** of variable is one of the following:
  - sc - scalar variable
  - ar(n) - array of size n

- **Intent** of variable is one of the following:
  - in - input only
  - out - output only
  - inout - both an input and an output variable

2.1.5. binini (Initializing Buffered Binary I/O Systems)

*deck,binini
  subroutine binini (iott)
  c *** primary function: initialize buffered binary i/o system
  c --- This routine is intended to be used in standalone programs.
  c --- This routine should not be linked into the ANSYS program.
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c   iott      (int,sc,in)       - output unit number for error output
  c output arguments:   none

2.1.6. Function sysiqr (Retrieving the Status of a File)

*deck,syisq
function sysiqr (nunit, fname, lname_in, inqr_in)

  c *** primary function: do a file system inquire (system dependent)
  c
  c  input arguments:
  c    variable (typ, siz, intent)    description
  c    nunit    (int, sc, in)       - fortran unit number (used only for inqr=''O'')
  c    fname    (chr, sc, in)       - name of file
  c    lname_in (int, sc, in)       - length of file name (characters, max=50)
  c    inqr_in  (chr, sc, in)       - character key for information requested
  c               = 'E' - return whether file exists
  c               = 1 - file exists
  c               = 0 - file does not exist
  c               < 0 - error occurred
  c               = 'O' - return whether file is open
  c               = 1 - file is open
  c               = 0 - file is closed
  c               < 0 - error occurred
  c               = 'N' - return unit number of file
  c               = > 0 - unit number for file
  c               = 0 - file not assigned to a unit
  c               < 0 - error occurred
  c
  c  output arguments:
  c    sysiqr   (int, func, out)    - the returned value of sysiqr is based on
  c    setting of inqr

2.1.7. Function biniqr8 (Retrieving System-Dependent Parameters)

*deck, biniqr8
  function biniqr8 (nblk, key)

  c *** primary function: get data about a block i/o buffer
  c --- This routine is intended to be used in standalone programs.
  c --- This routine should not be linked into the ANSYS program.
  c
  c  input arguments:
  c    nblk      (int, sc, in)       - the block number for the inquiry
  c                               or zero (see below)
  c    key       (int, sc, in)       - key for information requested
  c    nblk = 0 - return information about system/file
  c    key = 1 - return system block size
  c    = 2 - return number of integers per dp
  c    = 3 - return filename length
  c    = 5 - return integers per LONG
  c    nblk > 0 - return information about this block
  c    key = 1 - return fortran unit number
  c    = 2 - return number of blocks in file
  c    = 3 - return length of page (32 bit words)
  c    = 4 - return open status
  c    = 0 - file close
  c    = 1 - file open
  c    = 5 - return file format
  c    = 0 - internal format
  c    = 1 - external format
  c    = 6 - return read/write status
  c    = 0 - both read & write
  c    = 1 - read
  c    = 2 - write
  c    = 7 - return current position on file
  c    = 8 - return maximum length of file
  c    (in words)
  c    = 9 - return starting word for this page
  c    in buffer
  c
  c  output arguments:
  c    biniqr    (int, func, out)    - the returned value of biniqr is based on
2.1.8. Function binset (Opening a Blocked Binary File or Initializing Paging Space)

```fortran
*deck,binset
    function binset (nblk,nunit,ikeyrw,istart,paglen,npage,
    x  pname,nchar,kext,Buffer4)
     c *** primary function: initialize paging space for a blocked binary file.
     c     binset should be used to open a blocked file
     c     before binrd8 or binwrt8 are used. binclo should
     c     be used to close the file.
     c --- This routine is intended to be used in standalone programs.
     c --- This routine should not be linked into the ANSYS program.
     c *** Notice - This file contains ANSYS Confidential information ***
     c input arguments:
     c     nblk (int,sc,in) - block number (1 to BIO_MAXFILES max)
     c     nunit (int,sc,in) - fortran unit number for the file
     c     (if 0, bit bucket)
     c     ikeyrw (int,sc,in) - read/write flag
     c     =  0 - both read & write
     c     =  1 - read
     c     =  2 - write
     c     =  9 - read only
     c     istart (int,sc,in) - starting location in buffer array
     c     usually 1 for nblk=1, paglen*npage+1
     c     for nblk=2,etc.
     c     paglen (int,sc,in) - page length in integer*4 words for external
     c     files
     c     paglen should always be a multiple of
     c     512 words for efficiency
     c     npage (int,sc,in) - number of pages (1 to BIO_MAXBLOCKS max)
     c     pname (chr,ar(*),in) - name of the file
     c     nchar (int,sc,in) - number of characters in the file name(not
     c     used)
     c     kext (int,sc,in) - no longer used, always external format
     c     Buffer4 (i4, ar(*),inout) - work array for paging, should be
     c     dimensioned to paglen*npage*nblk (max)
     c output arguments:
     c     binset (int,func,out) - error status
     c     = 0 - no error
     c     <>0 - error occurred
     c     Buffer4 (i4, ar(*),inout) - work array for paging
```

2.1.9. Subroutine bintfo (Defining Data for a Standard ANSYS File Header)

```fortran
*deck,bintfo
    subroutine bintfo (title,jobnam,units,code)
     c *** primary function: set information necessary for binhed
     c --- This routine is intended to be used in standalone programs.
     c --- This routine should not be linked into the ANSYS program.
     c *** Notice - This file contains ANSYS Confidential information ***
     c input arguments:
     c     variable (typ,siz,intent) - description
     c     title (chr*80,ar(2),in) - main title and 1st subtitle
     c     jobnam (chr*8,sc,in) - jobname
     c     units (int,sc,in) - units
     c     = 0 - user defined units
     c     = 1 - SI (MKS)
     c     = 2 - CSG
```

---

Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.
**2.1.10. Subroutine binhed (Writing the Standard ANSYS File Header)**

*deck,binhed*

```fortran
subroutine binhed (nblk,nunit,filpos,buffer)
```

---

**input arguments:**
- `nblk` (int,sc,in) - block number of open binary file (as defined with subroutine binset)
- `nunit` (int,sc,in) - the unit number for this file
- `buffer` (int,ar(*),inout) - work array for paging, should be the same array as used in binset

**output arguments:**
- `filpos` (int,sc,out) - the position after the header
- `buffer` (int,ar(*),inout) - work array for paging

---

**loc no. words contents**
1  1 fortran unit number
2  2 file format
   = 0 - internal format
   = 1 - external format
3  1 time in compact form (ie 130619 is 13:06:19)
4  1 date in compact form (ie 19981023 is 10/23/1998)
5  1 units
   = 0 - user defined units
   = 1 - SI (MKS)
   = 2 - CSG
   = 3 - U.S. Customary, using feet
   = 4 - U.S. Customary, using inches
   = 6 - MPA
   = 7 - uMKS
6  1 User_Linked
10 1 revision in text format ' 5.0' (inexc4)
11 1 date of revision release for this version
12 3 machine identifier - 3 4-character strings
15 2 jobname - 2 4-character strings
17 2 product name - 2 4-character strings
19 1 special version label - 1 4-character string
20 3 user name - 3 4-character strings
23 3 machine identifier - 3 4-character strings
26 1 system record size at file write
27 1 maximum file length
28 1 maximum record number
31 8 jobname - 8 4-character strings
41 20 main title - 20 4-character strings
41 20 first subtitle - 20 4-character strings
95 1 split point of file
97-98 2 LONGINT of file size at write

---

**NOTE:** Split files are not support by binlib!
2.1.11. Subroutine binrd8 (Reading Data from a Buffered File)

*deck,binrd8
  subroutine binrd8 (nblk,LongLocL,leng,ivect,kbfint,Buffer4)
  c **********  buffer read routine   **********
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c nblk   (int,sc,in)     - block number.  see fd___(i.e. fdtri for tri
  c      (as defined with subroutine bioset)
  c LongLocL(LONG,sc,inout)- location in integer*4 words of the startin
  c      position on the file.
  c leng   (int,sc,inout)  - number of words to read into ivect. (must be
  c      less or equal to dimension given to ivect in
  c      the calling routine).  if ivect is to be used
  c      as integers, use as is.  if ivect is to be
  c      used for double precision numbers, it must be
  c      increased by multiplying it by INTPDP.
  c if negative, skip record and do not return
  c data(results).
  c Buffer4 (i4, ar(*),inout) - work array for paging, should be the
  c      same array as used in binset
  c output arguments:
  c LongLocL(LONG,sc,inout)- location in integer*4 words of the current
  c      position on the file.  It is updated after
  c      each read operation
  c leng   (int,sc,inout)  - tells you how many items it actually read(in
  c      integer words).
  c      if zero, end of file(error case)
  c ivect  (int,ar(*),out) - results (can be either integer or double
  c      precision in the calling routine)
  c kbfint (int,sc,out)    - key for type(used only for AUX2 dump)
  c      = 0  double precision data
  c      > 0  integer data(usually the same as leng)
  c Buffer4 (i4,ar(*),inout) - work array for paging

Versions of binrd8/binwrt8 exist without the "8" suffix (binrd/binwrt) that take a regular integer for the second argument. These subroutines, therefore, cannot address large files where the file position exceeds 2**31. Use the binrd8/binwrt8 versions for any new programs.

2.1.12. Subroutine binwrt8 (Writing Data to a Buffered File)

*deck,binwrt8
  subroutine binwrt8 (nblk,LongLocL,leng,ivect,kbfint,Buffer4)
  c primary function: buffer write routine
  c --- This routine is intended to be used in standalone programs.
  c --- This routine should not be linked into the ANSYS program.
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c nblk   (int,sc,in)    - block number.  see fd___(i.e. fdtri for tri
  c      (as defined with subroutine bioset)
  c LongLocL(LONG,sc,inout)- location in integer words of the starting
  c      position on the file.
  c leng   (int,sc,in)    - number of words to read from ivect. (must be
  c      less or equal to dimension given to ivect in
  c      the calling routine).  if ivect is to be used
  c      as integers, use as is.  if ivect is to be
  c      used for double precision numbers, it must be
  c      increased by multiplying it by INTPDP.
  c ivect  (int,ar(*),in) - data to be written onto the file(can be either
Versions of binrd8/binwrt8 exist without the "8" suffix (binrd/binwrt) that take a regular integer for the second argument. These subroutines, therefore, cannot address large files where the file position exceeds 2**31. Use the binrd8/binwrt8 versions for any new programs.

### 2.1.13. Subroutine exinc4 (Decoding an Integer String into a Character String)

```fortran
*deck, exinc4
subroutine exinc4 (ichext, chin, n)
  c primary function: decode externally formatted integer versions of 4-character strings to plain 4-character strings (used to convert data from externally formatted files to data for internally formatted files)
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
    c ichext    (int, ar(n), in) - externally formatted integer form of 4-character strings
    c n         (int, sc, in)       - number of strings to convert
  c output arguments:
    c chin      (char, ar(n), out) - strings in character form
```

### 2.1.14. Subroutine inexc4 (Coding a Character String into an Integer String)

```fortran
*deck, inexc4
subroutine inexc4 (chin, ichext, n)
  c primary function: encode plain 4-character strings into externally formatted integer versions of 4-character strings (used to convert data from internally formatted files to data for externally formatted files)
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
    c chin      (char, ar(n), in) - strings in character form
    c n         (int, sc, in)       - number of strings to convert
  c output arguments:
    c ichext    (int, ar(n), out) - externally formatted integer form of 4-character strings
```

### 2.1.15. Subroutine binclo (Closing or Deleting a Blocked Binary File)

```fortran
*deck, binclo
subroutine binclo (nblk, pstat, Buffer4)
  c *** primary function: close a blocked file, every block/file opened with
```
2.1.16. Subroutine largeIntGet (Converting Two Integers into a Pointer)

*deck,largeIntGet
function largeIntGet (small,large)
c primary function: Convert two short integers into a long integer
c object/library: res
c *** Notice - This file contains ANSYS Confidential information ***
c input arguments:
c   small   (int,sc,in)   - least significant part
c   large   (int,sc,in)   - most significant part
c output arguments:
c   largeIntGet  (LONGINT,sc,out)  - 64 bit integer

2.2. Demonstration Routines

The demonstration routines demonstrate several ways to use the binary file access routines provided with ANSYS. The programs described below (all available on your distribution media; see Section 2.1: Accessing ANSYS Binary Files for their location) demonstrate other tasks that the binary access routines can do.

2.2.1. Program bintst (Demonstrates Dumping a Binary File and Copying It for Comparison Purposes)

The bintst program dumps a binary file with the name file.rst to the screen. It then takes that file, copies it to a new file, file2.rst, and dumps the new file to the screen for comparison purposes.

2.2.1.1. Common Variables:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type, Size, Intent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iout</td>
<td>int, sc, comm</td>
<td>The output unit number</td>
</tr>
<tr>
<td>intpdp</td>
<td>int, sc, comm</td>
<td>The number of integers per double precision word</td>
</tr>
<tr>
<td>lenfrm</td>
<td>int, sc, comm</td>
<td>The number of characters in the filename</td>
</tr>
<tr>
<td>reclng</td>
<td>int, sc, comm</td>
<td>The system record length</td>
</tr>
</tbody>
</table>
Note

The bintst program is not part of the binlib.a library. It is included here only to aid you.

2.2.2. Subroutine bintrd (Demonstrates Printing a Dump of File Contents)

*deck, bintrd
  subroutine bintrd (pname)
  c *** primary function: bin file dump utility
  c
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  c *** ansys(r) copyright(c) 2000
  c *** ansys, inc.
  c
  typ=int, dp, log, chr, dcp  siz=sc, ar(n)  intent=in, out, inout
  c
  input arguments:
  c  variable  (typ, siz, intent)  description
  c  pname     (chr, sc, in)      - name of binary file which is to
  c                      be dumped to the screen
  c
  output arguments:
  c  none.
  c
  common variables:
  c  iout    (int, sc, comm)    - output unit number
  c  intpdp  (int, sc, comm)    - number of integers per double precision word
  c  lenfnm  (int, sc, comm)    - number of characters in the filename
  c  reclng  (int, sc, comm)    - system record length
  c
  NOTE: bintrd is not part of binlib.a. It is
  included only as an aid to users.

Note

The bintrd routine and the bintwr routine described below are not part of binlib.a. This
chapter includes it only to aid you. You can find the source for this routine on the ANSYS distribution
media.

Both subroutines require the following common:

COMMON/BINTCM/ IOUT, INTPDP, LENFNM, RECLNG

- Iout is the output unit number.
- Intpdp is the number of integers per double precision word.
- Lenfnm is the number of characters in the filename.
- Reclng is the system record length.

2.2.3. Subroutine bintwr (Demonstrates Copying Binary File Contents)

*deck, bintwr
  subroutine bintwr (pname, nname)
  c *** primary function: bin file copy utility
  c
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  c *** ansys(r) copyright(c) 2000
  c *** ansys, inc.
  c
  typ=int, dp, log, chr, dcp  siz=sc, ar(n)  intent=in, out, inout
2.2.4. Program wrtsub (Demonstrates Writing an ANSYS Substructure File)

*deck, wrtsub
  program wrtsub
  c primary function: demonstrates use of binary access routines
  c secondary function: write an ANSYS substructure file
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  c *** copyright(c) 2006 SAS IP, Inc. All rights reserved.
  c *** ansys, inc.

2.2.5. Program rdsubs (Demonstrates Reading a Substructure File)

Subroutine rdsubs demonstrates how you read an ANSYS substructure file. This demonstration program can handle up to \( \text{MAXNODE} \) nodes and \( \text{MAXDOF} \) degrees of freedom.

*deck, rdsubs
  program rdsubs
  c primary function: demonstrates use of binary access routines
  c secondary function: read an ANSYS substructure file
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  c *** copyright(c) 2006 SAS IP, Inc. All rights reserved.
  c *** ansys, inc.

2.2.6. Program rdfull (Demonstrates Reading and Reformatting the .FULL File)

Program rdfull demonstrates how to read and reformat the .FULL file. ANSYS writes the full file if the PSOLVE, ELFORM, PSOLVE, ELPREP, PSOLVE, TRIANG sequence is used. You can also use the WRFULL command.
If you want to use the free stiffness and mass matrices, make sure that there are no constraints on your model.

*deck,rdfull
  program rdfull
  c *** primary function: demonstrates use of binary access routines
  c *** secondary function: Read and reformat full file
  c
  c *** copyright(c) 2006 SAS IP, Inc. All rights reserved.
  c *** ansys, inc.
  c
  c NOTICE - A new assembly process, termed 'symbolic assembly', has
  c replaced the old assembly process, termed 'frontal
  c assembly', and is now the default assembly process for
  c most analyses. This program demonstrates how to read
  c and reformat the .FULL file that was created using
  c frontal assembly or symbolic assembly. ANSYS writes the
  c .FULL file if the PSOLVE,ELFORM
  c PSOLVE,ELPREP
  c PSOLVE,TRIANG
  c sequence is used. ANSYS will also write the .FULL file
  c when the sparse, ICCG, or JCG solver is used, as well as
  c when most mode extraction methods are used.
  c
  c Be sure to set up for modal ANTYPE,2
  c and full subspace MODOPT,SUBSP,nmode,0,0, ,OFF
  c (nmode is not used - it can be any value
  c
  c If the free-free stiffness and mass matrices are desired,
  c make sure there are no constraints on the model.

2.2.7. Program ResRdDemo (Demonstrates Reading a Results File)

Program ResRdDemo demonstrates how to read a results file using the results file access routines. The file must be named test.rst and the file contents are written to the screen.

This file resides in the subdirectory \Program Files\Ansys Inc\V110\ANSYS\custom\misc\<platform> (on Windows systems) or /ansys_inc/v110(ansys/customize/misc (on UNIX systems).

2.2.8. Program ResWrDemo (Demonstrates Writing a Results File)

Program ResWrDemo demonstrates how to write an ANSYS-readable results file. This file resides in the subdirectory \Program Files\Ansys Inc\V110\ANSYS\custom\misc\<platform> (on Windows systems) or /ansys_inc/v110(ansys/customize/misc (on UNIX systems).

2.3. Results File Access Routines

You can use the low-level routines in described in Section 2.1: Accessing ANSYS Binary Files to retrieve data from the results file. Alternatively, you can use the routines described in this section that retrieve the data specific to the format of the results file. These routines are included on the installation CD (compressed); you can install them on your system by performing a custom installation (as described in the ANSYS Installation and Configuration Guide for your platform).

These files reside in the subdirectory \Program Files\Ansys Inc\V110\ANSYS\custom\misc\<platform> (on Windows systems) or /ansys_inc/v110(ansys/customize/misc (on UNIX systems). See Section 2.1.1: Access Routines to Results and Substructure Files for information on compiling and linking these routines.

2.3.1. Overview of the Routines

For each data record in the results file, routines exist that:
• Read the record index and allocate space for the data. These are named `ResRdrecordBegin`, where `record` is a descriptive name of the record, e.g., `ResRdNodeBegin`

• Read the data itself. These are named `ResRdrecord`, e.g., `ResRdNode`

• Deallocate space for the data. These are named `ResRdrecordEnd`, e.g., `ResRdNodeEnd`

Below is a complete listing of all the routines with the indentation indicating the required nested calling sequence:

```plaintext
function ResRdBegin (Nunit, Lunit, Fname, ncFname, Title, JobName,
                     Units, NumDOF, DOF, UserCode,
                     MaxNode, NumNode, MaxElem, NumElem,
                     MaxResultSet, NumResultSet)  
```

These routines are contained in the file `ResRd.F`. See the demonstration routine `ResRdDemo.F` on the distribution medium for an example of the usage of these routines.

The memory allocation scheme is described in Memory Management Routines in the Guide to ANSYS User Programmable Features.

The following sections describe the data-reading routines. See the file `ResRd.F` and its corresponding include deck `ResRd.inc` for listings of the corresponding Begin/End routines.

2.3.2. ResRdBegin (Opening the File and Retrieving Global Information)

```plaintext
*deck,ResRdBegin  
function ResRdBegin (Nunit, Lunit, Fname, ncFname, Title, JobName,
                       Units, NumDOF, DOF, UserCode,
                       MaxNode, NumNode, MaxElem, NumElem,
                       MaxResultSet, NumResultSet)  
```

```plaintext
c primary function: Open result file and return global information
```
2.3.3. ResRdGeomBegin (Retrieving Global Geometry Information)

*deck,ResRdGeomBegin
   subroutine ResRdGeomBegin (MaxType, MaxReal, MaxCsys)
   c primary function:    Read Geometry Header Record

2.3.4. ResRdType (Retrieving Element Types)

*deck,ResRdType
   function ResRdType (itype,ielc)
   c primary function:    Read an element type record

2.3.5. ResRdReal (Retrieving Real Constants)

*deck,ResRdReal
   function ResRdReal (iReal,Rcon)
   c primary function:    Read real constant record
2.3.6. ResRdCsyst (Retrieving Coordinate Systems)

*deck,ResRdCsyst
  function ResRdCsyst (iCsys,Csys)
  c primary function: Read a coordinate system record
  c object/library: ResRd

  c input arguments:
  c   iCsys   (int,sc,in) - Coordinate system number
  c
  c output arguments:
  c   Csys    (dp,ar(ResRdCsyst),out)- Coordinate system description
  c   ResRdCsyst (int,sc,out) - Number of values

2.3.7. ResRdNode (Retrieving Nodal Coordinates)

*deck,ResRdNode
  function ResRdNode (iNode,xyzang)
  c primary function: Get a node
  c object/library: ResRd

  c input arguments:
  c   iNode    (int,sc,in)   - node sequence number
  c                                                 (1 to NumNode)
  c
  c output arguments:
  c   xyzang   (dp,ar(6),out) - x,y,z,thxy,thyz,thzx for node
  c   ResRdNode (int,sc,out) - Node number

2.3.8. ResRdElem (Retrieving Elements)

*deck,ResRdElem
  function ResRdElem (iElem, nodes, ElemData)
  c primary function: Read an element

  c object/library: ResRd

  c input arguments:
  c   iElem    (int,sc,in) - The element number
  c
  c output arguments:
  c   ResRdElem(int,sc,out) - Number of nodes
  c   nodes    (int,ar(n),out) - Element nodes
  c   ElemData (int,ar(10),out) - Element information
  c                         mat - material reference number
  c                         type - element type number
  c                         real - real constant reference number
  c                         secnum - section number
  c                         esys - element coordinate system
  c                         death - death flag
  c                         = 0 - alive
  c                         = 1 - dead
  c                         solidm - solid model reference
  c                         shape - coded shape key
  c                         elnum - element number
  c                         pexcl - P-Method exclude key
2.3.9. ResRdSolBegin (Retrieving Result Set Location)

*deck, ResRdSolBegin
  function ResRdSolBegin (key,lstep,substep,ncumit,kcmplx,time, 
  x                             Title,DofLab)
 c primary function: Read the solution header records

 c object/library: ResRd

 c input arguments:
 c   key   (int,sc,in)   - 1, find by lstep/substep
 c   lstep (int,sc,in/out) - Load step number
 c   substep (int,sc,in/out) - Substep of this load step
 c   ncumit (int,sc,in/out) - Cumulative iteration number
 c   kcmplx (int,sc,in)   - 0, Real solution   1, Imaginary solution
 c   time  (int,sc,in/out) - Current solution time

 c output arguments:
 c   Title  (ch*80,ar(5),out)  - Title and 4 subtitles
 c   DofLab (ch*4,ar(nDOF),out) - Labels for DOFs
 c   ResRdSolBegin (int,sc,out) - 0, requested solution set found
 c                              1, not found

2.3.10. ResRdDisp (Retrieving Nodal Solution)

*deck, ResRdDisp
  function ResRdDisp (node,Disp)
 c primary function: Retrieve a nodal displacement

 c object/library: ResRd

 c input arguments:
 c   node  (int,sc,in)   - Node number

 c output arguments: none
 c   Disp  (dp,ar(nDOF),out) - Displacements
 c   ResRdDisp(int,sc,out)  - Number of displacements

2.3.11. ResRdRfor (Retrieving Reaction Solution)

*deck, ResRdRfor
  function ResRdRfor (node,idof,value)
 c primary function: Retrieve a reaction force

 c object/library: ResRd

 c input arguments:
 c   node  (int,sc,in)   - External node number
 c   idof  (int,sc,in)   - Internal dof number

 c output arguments:
 c   value  (dp,sc,in)   - Value of reaction force
 c   ResRdRfor (int,sc,out)  - Number of returned values (0 or 1)

2.3.12. ResRdFix (Retrieving Applied Nodal Constraints)

*deck, ResRdFix
  function ResRdFix (node,idof,value)
 c primary function: Retrieve a constraint value

 c object/library: ResRd

 c input arguments:
 c   node  (int,sc,in)   - External node number
2.3.13. ResRdForc (Retrieving Applied Nodal Loads Solution)

*deck,ResRdForc
   function ResRdForc (node,idof,value)
   c primary function: Retrieve an applied force value
   c object/library: ResRd

   c input arguments:
   c   node   (int,sc,in)         - External node number
   c   idof   (int,sc,in)         - Internal dof number

   c output arguments:
   c   value   (dp,ar(4),in)     - Real, Imag, RealOld, ImagOld
   c   ResRdForc (int,sc,out)    - Number of returned values (0 or 4)

2.3.14. ResRdEstr (Retrieving Element Solutions)

*deck,ResRdEstr
   function ResRdEstr (iStr,Str)
   c primary function: Get an element’s results
   c object/library: ResRd

   c input arguments:
   c   iStr   (int,sc,in)         - element record number (1-25)

   c output arguments:
   c   ResRdEstr (int,sc,out)    - Number of element values
   c   Str     (dp,ar(nStr),out) - element values
Chapter 3: Using CDREAD and CDWRITE

3.1. Using the CDREAD Command

The **CDREAD** command and its GUI equivalent, **Main Menu> Preprocessor> Archive Model> Read**, read a file of model and database information into the ANSYS database. The commands and menu paths listed below define the input data that the ANSYS program requires to solve a model. If the file you are reading into the database via **CDREAD** or **Main Menu> Preprocessor> Archive Model> Read** does not contain all of the required input data, you can use these commands or menu paths to define that data in the preprocessor. For detailed information about the commands, see the *Commands Reference*.

In the following list, commands or menu paths shown with an asterisk (*) are the minimum requirements for a solution.

<table>
<thead>
<tr>
<th>Command</th>
<th>Equivalent Menu Path</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>/PREP7*</td>
<td>Main Menu&gt; Preprocessor*</td>
<td>Enters the general preprocessor</td>
</tr>
<tr>
<td>ET*</td>
<td>Main Menu&gt; Preprocessor&gt; Element Type&gt; Add/Edit/Delete*</td>
<td>Defines the element types</td>
</tr>
<tr>
<td>MP*</td>
<td>Main Menu&gt; Preprocessor&gt; Material Props*</td>
<td>Defines the material properties</td>
</tr>
<tr>
<td>MPTEMP</td>
<td>Main Menu&gt; Preprocessor&gt; Material Props&gt; Material Models</td>
<td>Defines a table of material temperatures</td>
</tr>
<tr>
<td>MPDATA</td>
<td>Main Menu&gt; Preprocessor&gt; Material Props&gt; Material Models</td>
<td>Defines a table of material properties</td>
</tr>
<tr>
<td>TB</td>
<td>Main Menu&gt; Preprocessor&gt; Material Props&gt; Material Models</td>
<td>Defines nonlinear material data, some element data, or both</td>
</tr>
<tr>
<td>R</td>
<td>Main Menu&gt; Preprocessor&gt; Real Constants&gt; Add/Edit/Delete*</td>
<td>Defines element real constants</td>
</tr>
<tr>
<td>LOCAL</td>
<td>Utility Menu&gt; WorkPlane&gt; Local Coordinate Systems&gt; Create Local CS&gt; At Specified Loc</td>
<td>Defines a local coordinate system</td>
</tr>
<tr>
<td>N*</td>
<td>Main Menu&gt; Preprocessor&gt; Modeling&gt; Create&gt; Nodes&gt; In Active CS or Main Menu&gt; Preprocessor&gt; Modeling&gt; Create&gt; Nodes On Working Plane</td>
<td>Defines a nodal location</td>
</tr>
<tr>
<td>E or EN*</td>
<td>Main Menu&gt; Preprocessor&gt; Modeling&gt; Create&gt; Elements&gt; Auto Numbered&gt; Thru Nodes or Main Menu&gt; Preprocessor&gt; Modeling&gt; Create&gt; Elements&gt; User Numbered&gt; Thru Nodes</td>
<td>Defines an element. (You must set the correct TYPE, MAT, REAL, and ESYS pointers before defining the element.)</td>
</tr>
<tr>
<td>ANTYPE</td>
<td>Main Menu&gt; Preprocessor&gt; Loads&gt; Analysis Type&gt; New Analysis</td>
<td>Defines the analysis type. See the <em>Commands Reference</em> for details.</td>
</tr>
<tr>
<td>M</td>
<td>Main Menu&gt; Preprocessor&gt; Loads&gt; Master DOFs&gt; User Selected&gt; Define</td>
<td>Defines master degrees of freedom</td>
</tr>
<tr>
<td>ACCEL</td>
<td>Main Menu&gt; Preprocessor&gt; Loads&gt; Define Loads&gt; Apply&gt; Structural&gt; Inertia&gt; Gravity</td>
<td>Defines the acceleration vector</td>
</tr>
<tr>
<td>D*</td>
<td>Main Menu&gt; Preprocessor&gt; Loads&gt; Define Loads&gt; Apply&gt; constraint</td>
<td>Defines degree of freedom constraints</td>
</tr>
<tr>
<td>F</td>
<td>Main Menu&gt; Preprocessor&gt; Loads&gt; Define Loads&gt; Apply&gt; force type&gt; On entity</td>
<td>Defines nodal forces</td>
</tr>
</tbody>
</table>
### 3.1.1. Tips for Reading Files with CDREAD

The following list describes practices to follow when reading files with **CDREAD** or its menu path equivalent:

- If you place the command */NOPR* at the beginning of the file and */GOPR* at the end of the file, you can suppress printing of the file and speed up the time to read it.

- Use the */COM* command to add comments to the file. If the file contains the */NOPR* command, */COM* provides status information about what is being read in. For example:

  ```
  /COM, READING NODES
  N,1,.....
  ,
  ,
  /COM, READING ELEMENTS
  EN,1,.....
  ```

- Group elements with the same attributes (TYPE, REAL, MAT, ESYS). *Do not* use a separate MAT, REAL, TYPE, and ESYS command for each element.

- Use **Utility Menu > File > Change Title** or the */TITLE* command to add a title to graphics displays and printouts.

- Use **Utility Menu > File > Read Input From** or the */INPUT* command at the Begin level to input the file.

### 3.2. Using the CDWRITE Command

To export a model from the ANSYS program to another application, use menu path **Main Menu > Preprocessor > Archive Model > Write** or the CDWRITE command within the general preprocessor, PREP7. This produces a coded database file called *Jobname.cdb*. You specify the jobname using **Utility Menu > File > Change Jobname** or the */FILNAM* command. If you supply no jobname, the ANSYS program uses the default name "file".

The *Jobname.cdb* file contains selected geometry (nodes and elements), load items, and other model data in terms of ANSYS input commands. (For a complete list of data in the file, see the CDWRITE description in the **Commands Reference**.) You can convert this information to a format compatible with the program into which you are importing it. The next few pages describe special considerations and commands you may need to do this conversion.

**Note**

Files created by the CDWRITE command have the active coordinate system set to Cartesian (**CSYS, 0**).

ANSYS may create parameters in the CDWRITE file that start with an underscore (_), usually an "_z." Such parameters are for ANSYS internal use and pass information to the ANSYS GUI.
3.2.1. Customizing Degree of Freedom Labels: the /DFLAB Command

The ANSYS program uses a set of default labels for the degrees of freedom. You use these labels when entering boundary conditions, or ANSYS uses the labels when writing the Jobname.cdb file.

You can change the labels to reflect the degrees of freedom of the other program by issuing the command /DFLAB. If you are customizing the DOF labels, /DFLAB must be the first command you enter within the ANSYS program. You may want to include the command in your START.ANS file. You can use /DFLAB only at the Begin processing level.

/DFLAB assigns or reassigns the "displacement" and "force" labels in the ANSYS DOF list. For example, degree of number 1 is predefined to have a displacement label of UX and a force label of FX, but you can assign new labels to this DOF using by issuing /DFLAB. Changing predefined labels generates a warning message.

The format for the /DFLAB command is:

/DFLAB, NDOF, LabD, LabF

NDOF
ANSYS degree of freedom number (1 to 32)

LabD
Displacement degree of freedom label to be assigned (up to four characters)

LabF
Force label to be assigned (up to four characters)

You can also use /DFLAB to assign labels to spare degree of freedom numbers. Spare displacement and force labels are from 13 to 18 and from 27 to 32. All other DOF numbers are predefined, as follows:

<table>
<thead>
<tr>
<th>DOF Number</th>
<th>Corresponding Displacement Label</th>
<th>Corresponding Force Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>UX</td>
<td>FX</td>
</tr>
<tr>
<td>2</td>
<td>UY</td>
<td>FY</td>
</tr>
<tr>
<td>3</td>
<td>UZ</td>
<td>FZ</td>
</tr>
<tr>
<td>4</td>
<td>ROTX</td>
<td>MX</td>
</tr>
<tr>
<td>5</td>
<td>ROTY</td>
<td>MY</td>
</tr>
<tr>
<td>6</td>
<td>ROTZ</td>
<td>MZ</td>
</tr>
<tr>
<td>7</td>
<td>AX</td>
<td>CSGX</td>
</tr>
<tr>
<td>8</td>
<td>AY</td>
<td>CSGY</td>
</tr>
<tr>
<td>9</td>
<td>AZ</td>
<td>CSGZ</td>
</tr>
<tr>
<td>10</td>
<td>VX</td>
<td>VFX</td>
</tr>
<tr>
<td>11</td>
<td>VY</td>
<td>VFY</td>
</tr>
<tr>
<td>12</td>
<td>VZ</td>
<td>VFZ</td>
</tr>
<tr>
<td>13</td>
<td>PRES</td>
<td>FLOW</td>
</tr>
<tr>
<td>14</td>
<td>TEMP</td>
<td>HEAT</td>
</tr>
<tr>
<td>15</td>
<td>VOLT</td>
<td>AMPS</td>
</tr>
<tr>
<td>16</td>
<td>MAG</td>
<td>FLUX</td>
</tr>
<tr>
<td>17</td>
<td>ENKE</td>
<td>NPKE</td>
</tr>
<tr>
<td>18</td>
<td>ENDS</td>
<td>NPDS</td>
</tr>
<tr>
<td>19</td>
<td>EMF</td>
<td>CURT</td>
</tr>
<tr>
<td>20</td>
<td>CURR</td>
<td>VLTG</td>
</tr>
</tbody>
</table>
3.3. Coded Database File Commands

In the coded database file Jobname.CDB, most ANSYS commands have the same format they have elsewhere. (See the Commands Reference for command-specific information.) However, the format for some commands differs slightly in the Jobname.CDB file. The format for these commands is described below.

The CDWRITE command has an UNBLOCKED and a BLOCKED option. The UNBLOCKED option will write all data out in command format, the default BLOCKED option will write certain data items in a fixed format, especially those which could potentially contain large amounts of data, such as nodal data.

3.3.1. CE Command

The CE command defines the constant term in a constraint equation. The command format in Jobname.CDB is:

```
CE, R5.0, Type, LENGTH, NCE, CONST
```

**Type**
- The type of data to be defined. DEFI is the valid label.

**LENGTH**
- The total number of variable terms in the constraint equation.

**NCE**
- The constraint equation reference number.

**CONST**
- The constant term of the equation.

Another version of the CE command defines the variable terms in a constraint equation. You must issue this version of the command after the CE command described above. This command repeats until all terms are defined.

The alternate format for the CE command is:

```
CE, R5.0, Type, N1, Dlab1, C1, N2, Dlab2, C2
```

**Type**
- The type of data to be defined. NODE is the valid label.

**N1**
- The node number of the next term.

**Dlab1**
- The DOF label of N1.

**C1**
- The coefficient of N1.

**N2**
- The node number of the next term.

**Dlab2**
- The DOF label of N2.

**C2**
- The coefficient of N2.
3.3.2. CP Command

The CP command defines a coupled node set. You repeat the command until all nodes are defined. The command format in Jobname.CDB is:

```
CP, R5.0, LENGTH, NCP, Dlab, N1, N2, N3, N4, N5, N6, N7
```

**LENGTH**
- The total number of nodes in the coupled set

**NCP**
- The coupled node reference number

**Dlab**
- The degree of freedom label for the set

**N1, N2, N3, N4, N5, N6, N7**
- The next seven node numbers in the coupled set

3.3.3. CMBLOCK Command

The CMBLOCK command defines the entities contained in a node or element component. The command format in Jobname.CDB is:

```
CMBLOCK, Cname, Entity, NUMITEMS
```

**Cname**
- Eight character component name.

**Entity**
- Label identifying the type of component (NODE or ELEMENT).

**NUMITEMS**
- Number of items written.

**Format**
- Data descriptors defining the format. For CMBLOCK this is always (8i10).

The items contained in this component are written at 10 items per line. Additional lines are repeated as needed until all NumItems are defined. If one of the items is less than zero, then the entities from the item previous to this one (inclusive) are part of the component.

3.3.4. EBLOCK Command

The EBLOCK command defines a block of elements. The command syntax is:

```
EBLOCK, NUM_NODES, Solkey
```

**NUM_NODES**
- The number of nodes to be read in the first line of an element definition.

**Solkey**
- The solid model key. The element is part of a solid model if the keyword SOLID appears here. When Solkey = SOLID, Field 8 (the element shape flag) may be left at zero, and Field 9 is the number of nodes defining this element.

**Format**
- Data descriptors defining the format. This must be 19i7.
The format of the element "block" is as follows:

- Field 1 - The material number.
- Field 2 - The element type number.
- Field 3 - The real constant number.
- Field 4 - The section ID attribute (beam section) number.
- Field 5 - The element coordinate system number.
- Field 6 - The birth/death flag.
- Field 7 - The solid model reference number.
- Field 8 - The element shape flag.
- Field 9 - The number of nodes defining this element if Solkey = SOLID; otherwise, Field 9 = 0.
- Field 10 - The exclude key (p-elements).
- Field 11 - The element number.
- Fields 12-18 - The node numbers. The next line will have the additional node numbers if there are more than eight.

The final line of the block will be a -1 in field 1.

If you are in the GUI, the EBLOCK command must be contained in an externally prepared file and read into ANSYS (i.e., CDREAD, /INPUT, etc.).

3.3.5. EDCADAPT Command

The EDCADAPT command specifies adaptive meshing control for explicit dynamics analysis. The command format in Jobname.CDB is:

```
EDCADAPT, R5.3, FREQ, TOL, OPT, MAXLVL, BTIME, DTIME, LCID, ADPSIZE, ADPASS, IREFLG, ADPENE, ADPTH, MAXEL
```

**FREQ**
- The time interval between adaptive mesh refinement.

**TOL**
- The adaptive angle tolerance (in degrees).

**OPT**
- The adaptivity option.

**MAXLVL**
- The maximum number of mesh refinement levels.

**BTIME**
- The birth time to begin adaptive meshing.

**DTIME**
- The death time to end adaptive meshing.

**LCID**
- The curve ID defined by EDCURVE

**ADPSIZE**
- The minimum element size to be adapted, based on the element edge length.

**ADPASS**
- The one-pass or two-pass adaptivity option.
IREFLG
The uniform refinement level flag.

ADPENE
Adaptive mesh flag for starting adaptivity when approaching (positive ADPENE) or penetrating (negative ADPENE) the tooling surface.

ADPTH
Absolute shell thickness level below which adaptivity should begin.

MAXEL
The maximum number of elements at which adaptivity will be terminated.

NOTE: This command is also listed in the Commands Reference. The format listed here contains information specific to the CDREAD/CDWRITE file.

3.3.6. EDCGEN Command

The EDCGEN command is used to define a contact definition for explicit dynamics. The command format in Jobname.CDB is:

EDCGEN, R5.3, Option, Cont, Targ, Lkey, FS, FD, DC, VC, VDC, V1, V2, V3, V4, BTIME, DTIME, BOXID1, BOXID2

Option
The label identifying the contact behavior.

Cont
The contact surface, identified by component name, part ID, or part assembly ID.

Targ
The target surface, identified by component name, part ID, or part assembly ID.

Lkey
A key identifying the meaning of Cont and Targ (component, part or part assembly).

FS
The static friction coefficient.

FD
The dynamic friction coefficient.

DC
The exponential decay coefficient.

VC
The coefficient of viscous friction.

VDC
The viscous damping coefficient in percent of critical damping.

V1, V2, V3, V4
Additional input for some contact types. See EDCGEN in the Commands Reference for more information.

BTIME
The birth time for which contact definition will become active.

DTIME
The death time for which contact definition will become inactive.

BOXID1
Contact volume as defined using EDBOX
Target volume as defined using EDBOX

NOTE: This command is also listed in the Commands Reference. The format listed here contains information specific to the CDREAD/CDWRITE file.

3.3.7. EDCURVE Command

The EDCURVE command is used to define a curve for an explicit dynamics analysis. The command format in Jobname.CDB is:

EDCURVE, R5.3, Option, LCID, Length, 0.0, Par1, Par2

Option
The EDCURVE command option. The only valid option is “ADD.”

LCID
The curve ID.

Length
The number of data values for the abscissa array (Par1) and the ordinate array (Par2).

Par1
The abscissa values, repeat Length number of times.

Par2
The ordinate values, repeat Length number of times.

NOTE: This command is also listed in the Commands Reference. The format listed here contains information specific to the CDREAD/CDWRITE file.

3.3.8. EDDRELAX Command

The EDDRELAX command activates initialization to a prescribed geometry or dynamic relaxation for the explicit analysis. The command format in Jobname.CDB is:

EDDRELAX, R5.4, Option, NRCYCK, IRELAL, EDTTL, DRTOL, DFFCTR, DRTERM, TSSFDR

Option
The EDDRELAX command option. Valid options are “ANSYS” (relaxation is based on the implicit analysis, see the EDDRELAX command in the Commands Reference) or “DYNA,” where the relaxation parameters are controlled within the LS-DYNA analysis. The following arguments are valid for Option = DYNA only.

NRCYCK
The number of iterations between the convergence checks.

IRELAL
Automatic control based on Papadrakakis not active (0) or active (1).

EDTTL
The convergence tolerance when automatic control is used.

DRTOL
The convergence tolerance.

DFFCTR
The dynamic relaxation factor.

DRTERM
The termination time for dynamic relaxation.
3.3.9. EDLCS Command

The EDLCS command is used to define a local coordinate system for explicit dynamics. The command format in Jobname.CDB is:

```
EDLCS,R5.3,Option,CID,X1,Y1,Z1,X2,Y2,Z2,X3,Y3,Z3
```

**Option**
- The EDLCS command option. The only valid option is “ADD.”

**CID**
- The coordinate system ID.

**X1,Y1,Z1**
- The X,Y,Z coordinate of a point on the local X-axis.

**X2,Y2,Z2**
- The X,Y,Z coordinate of a point on the local X-Y plane.

**X3,Y3,Z3**
- The X,Y,Z coordinate of the local origin.

NOTE: This command is also listed in the Commands Reference. The format listed here contains information specific to the CDREAD/CDWRITE file.

3.3.10. EDLOAD Command

The EDLOAD command is used to define loading conditions for explicit dynamics. The command format in Jobname.CDB is:

```
EDLOAD,R5.3,Option,Lab,KEY,Cname,Length,PHASE,Par1,Par2,LCID,SCALE,BTIME,DTIME
```

**Option**
- The EDLOAD command option. The only valid option is “ADD.”

**Lab**
- The load labels.

**Key**
- The coordinate system number defined by EDLCS or the element face number for the pressure loading.

**Cname**
- The name of the existing component or part number to which this load will be applied.

**Length**
- The number of data values for the time array (Par1) and the load array (Par2).

**PHASE**
- Phase of the analysis in which the load curve is to be used.

**Par1,Par2**
- The time values, with the number of values in the string defined by the Length argument (above).

**Par2**
- The load values, with the number of values in the string defined by the Length argument (above).
LCID
The curve ID, created using the EDCURVE command. If LCID is nonzero, then Length= 1, and Par1 and Par2 will be equal to 0.

Scale
The Scale Factor applied to the load curve.

Btime
The birth time.

Dtime
The death time.

NOTE: This command is also listed in the Commands Reference. The format listed here contains information specific to the CDREAD/CDWRITE file.

3.3.11. EDPREAD Command

The EDPREAD command is used to internally write the part information to the Jobname.CDB file for explicit dynamics. Prior to Release 8.0, the command format in Jobname.CDB is:

EDPREAD,R5.4,Nmat,Npart
Type, Mat, Real, Used

Nmat
The number of materials.

Npart
Number of parts, and also, the number of times to repeat the second Type, Mat, Real, Used input line.

Type
The element type number.

Mat
The material number.

Real
The real constant set number.

Used
The flag indicating if the part is used (1), or not used (0).

For Release 8.0 and beyond, the command format is:

EDPREAD,R8.0,Nmat,Npart,Part ID
Type, Mat, Real, Used

Nmat
The number of materials.

Npart
Number of parts, and also, the number of times to repeat the second Type, Mat, Real, Used input line.

Part ID
The part number.

Type
The element type number.

Mat
The material number.
3.3.12. EDWELD Command

The EDWELD command is used to define a spotweld or a generalized weld for explicit dynamics.

There are two command formats (for spot and generalized welds). The command format for the spotweld appears in Jobname.CDB as follows:

```
EDWELD, R5.3, Option, NWELD, N1, N2, SN, SS, EXPN, EXPS
```

**Option**
The EDWELD command option. The only valid option is “ADD.”

**NWELD**
The spotweld ID number.

**N1**
The node number of the first node connected by the spotweld.

**N2**
The node number of the second node connected by the spotweld.

**SN**
The normal force at the spotweld failure.

**SS**
The shear force at the spotweld failure.

**EXPN**
The exponent for spotweld normal force.

**EXPS**
The exponent for spotweld shear force.

The command format for the generalized weld appears in Jobname.CDB as follows:

```
EDWELD, R5.3, Option, NWELD, CNAME, SN, SS, EXPN, EXPS, EPSF, TFAIL, NSW, CID
```

**Option**
The EDWELD command option. The only valid option is “ADD.”

**NWELD**
The generalized weld ID number.

**CNAME**
The name of the node component.

**SN**
The normal force at the weld failure.

**SS**
The shear force at the weld failure.

**EXPN**
The exponent for weld normal force.
EXPS
The exponent for weld shear force.

EXPF
The effective plastic strain at ductile failure.

TFAIL
The time of failure of the weld.

NSW
The number of spotwelds for the generalized weld.

CID
The coordinate system ID as defined by the EDLCS command.

NOTE: This command is also listed in the Commands Reference. The format listed here contains information specific to the CDREAD/CDWRITE file.

3.3.13. EN Command

The EN command is used to define an element. If an element contains more than eight nodes, the EN command is repeated until all nodes are defined. The command format in Jobname.CDB is:

```
EN, R5.5, Type, NUMN, I1, I2, I3, I4, I5, I6, I7, I8
```

The type of data to be defined. Valid labels are “ATTR” (read in element attributes), and “NODE” (read in nodes defining the element).

NUMN
The number of nodes.

I1, I2, I3, I4, I5, I6, I7, I8
The integer values to be read:

- If Type is ATTR, the integer values are the element attributes. Attributes are in the order: NUMN, MAT, TYPE, REAL, SECNUM, ESYS, NUMELEM, SOLID, DEATH, EXCLUDE
- If Type is NODE, the integer values are the node numbers.

3.3.14. LOCAL Command

The LOCAL command defines a local coordinate system. The command format in Jobname.CDB is:

```
LOCAL, R5.0, Type, NCSY, CSYTyp, VAL1, VAL2, VAL3
```

Type
The type of data to be defined. Valid labels are LOC (read in system origin), ANG (read in rotation angles), and PRM (read in system parameters).

NCSY
The coordinate system reference number.

CSYTyp
The coordinate system type (0, 1, 2, or 3).

VAL1, VAL2, VAL3
Values to be read:

- If Type is LOC, values are the system origin in global Cartesian coordinates.
- If Type is ANG, values are the rotation angles in degrees.
- If Type is PRM, values are the first and second parameters of the system.
3.3.15. M Command

The M command defines a master degree of freedom. The command format in Jobname.CDB is:

```
M, R5.0, NODE, Dlab
```

- **NODE**
  - The node number
- **Dlab**
  - The DOF label

3.3.16. MPDATA Command

The MPDATA command defines a material property data table. You repeat the command until all properties are defined. The command format in Jobname.CDB is:

```
MPDATA, R5.0, LENGTH, Lab, MAT, STLOC, VAL1, VAL2, VAL3
```

- **LENGTH**
  - The total number of temperatures in the table.
- **Lab**
  - The material property label. See the MP command description in Commands Reference for valid labels.
- **MAT**
  - The material reference number.
- **STLOC**
  - The starting location in the table for the next three property values.
- **VAL1, VAL2, VAL3**
  - Property values assigned to three locations in the table starting at STLOC.

3.3.17. MPTEMP Command

The MPTEMP command defines a temperature table. You repeat the command until all temperature values are defined. The command format in Jobname.CDB is:

```
MPTEMP, R5.0, LENGTH, STLOC, TEMP1, TEMP2, TEMP3
```

- **LENGTH**
  - The total number of temperatures in the table
- **STLOC**
  - The starting location in the table for the next three temperature values.
- **TEMP1, TEMP2, TEMP3**
  - Temperatures assigned to three locations in the table starting at STLOC.

3.3.18. N Command

If the UNBLOCKED option is used with the CDWRITE command, then the N command defines a node. This is also the method used for defining nodes in .CDB files before ANSYS 5.4. The command format in Jobname.CDB is:

```
N, R5.0, Type, NODE, SOLID, PARM, VAL1, VAL2, VAL3
```

- **Type**
  - The type of data to be defined. Valid labels are LOC (read in coordinates) and ANG (read in rotation angles).
3.3.19. NBLOCK Command

The NBLOCK command defines a block of nodes. This is the recommended method for inputting nodes into the ANSYS data base. The command syntax is:

```
NBLOCK, NUMFIELD, Solkey
```

**NUMFIELD**

The number of fields in the blocked format.

**Solkey**

The solid model key. The node is part of a solid model if the keyword SOLID appears here.

**Format**

Data descriptors defining the format. This must be (3i8,6e16.9).

The format of the node "block" is as follows:

- Field 1 - Node number.
- Field 2 - The solid model entity (if any) in which the node exists.
- Field 3 - The line location (if the node exists on a line).
- Field 4 - 6 - The nodal coordinates.
- Field 7 - 9 - The rotation angles.

Only the last nonzero coordinate/rotation is output; any trailing zero values are left blank.

The final line of the block is always an N command using a -1 for the node number.

The following example shows a typical NBLOCK formatted set of node information. Note that this example has no rotational data. It contains only the first six fields.

```
NBLOCK, 6
     (i8, 6e16, 9)
     1   6.21299982  0.625999987 -1.019883480E-07
     2   6.14472103  0.625999987  0.156284466
     3   6.21271753  0.625999987  1.096193120E-02
     .
     .
     151464 0 0 5.85640764 -0.442010075  1.911501959E-02
     151465 0 0 5.88715029 -0.442010075  7.201258256E-08
     151466 0 0 5.85541868 -0.442010075  7.201258256E-08
     N, R5.3, LOC, -1
```
If you are in the GUI, the **NBLOCK** command must be contained in an externally prepared file and read into ANSYS (i.e., **CDREAD**, /**INPUT**, etc.).

### 3.3.20. R Command

The **R** command defines a real constant set. You repeat the command until all real constants for this set are defined. The command format in Jobname.CDB is:

\[
R, R5.0, NSET, Type, STLOC, VAL1, VAL2, VAL3
\]

**NSET**
The real constant set reference number.

**Type**
The type of data to be defined. LOC is the valid label.

**STLOC**
The starting location in the table for the next three constants.

**VAL1**, **VAL2**, **VAL3**
Real constant values assigned to three locations in the table starting at **STLOC**.

### 3.3.21. RLBLOCK Command

The **RLBLOCK** command defines a real constant set. The real constant sets follow each set, starting with Format1 and followed by one or more Format2's, as needed. The command format is:

\[
RLBLOCK, NUMSETS, MAXSET, MAXITEMS, NPERLINE
\]

**NUMSETS**
The number of real constant sets defined

**MAXSET**
Maximum real constant set number

**MAXITEMS**
Maximum number of reals in any one set

**NPERLINE**
Number of reals defined on a line

**Format1**
Data descriptor defining the format of the first line. For the **RLBLOCK** command, this is always (2i8,6g16.9). The first i8 is the set number, the second i8 is the number of values in this set, followed by up to 6 real constant values.

**Format2**
Data descriptors defining the format of the subsequent lines (as needed); this is always (7g16.9).

The real constant sets follow, with each set starting with Format1, and followed by one or more Format2's as needed.

### 3.3.22. SECBLOCK Command

The **SECBLOCK** command retrieves all mesh data for a user-defined beam section as a block of data. You repeat the command for each beam section that you want to read. The command format is:

\[
SECBLOCK
\]

**Format1**
Data descriptor defining the format of the first line.
The First Line section. The first value is the number of nodes, and the second is the number of cells.

The Cells Section. The first 9 values are the cell connectivity nodes. The 10th (last) value is the material ID (MAT).

The Nodes Section. This section contains as many lines as there are nodes. In this example, there are 27 nodes, so a total of 27 lines would appear in this section. Each node line contains the node's boundary flag, the Y coordinate of the node, and the Z coordinate of the node. Currently, all node boundary flags appear as 0s in a cell mesh file. Because all node boundary flags are 0, SECBLOCK ignores them when it reads a cell mesh file.

Sample User Section Cell Mesh File

Following is a sample excerpt from a custom section mesh file for a section with 27 nodes, 4 cells, and 9 nodes per cell:

**First Line:**

```
27   4
```

**Cells Section:**

```
1   3   11   9   2   6   10   4   5   2
7   9   23   21   8   16   22   14   15   1
9   11  25   23   10  18   24   16   17   1
11  13  27   25   12  20   26   18   19   1
```

**Nodes Section:**

```
0   0.0          0.0
0   0.025        0.0
0   0.05         0.0
0   5.0175       0.0
0   19.98        10.00
0   20.00        10.00
... 
```

3.3.23. SFBEAM Command

The **SFBEAM** command defines a surface load on selected beam elements. Remaining values associated with this specification are on a new input line with a (4f16.9) format. The command format in Jobname.CDB is:

```
SFBEAM, ELEM, LKEY, Lab, R5.0, DIOFFST, DJOFFST
```

**ELEM**

The element number.

**LKEY**

The load key associated with these surface loads.

**Lab**

A label indicating the type of surface load. PRES (for pressure) is the only valid label.

**DIOFFST**

Offset distance from node I.

**DJOFFST**

Offset distance from node J.
3.3.24. SFE Command

The SFE command defines a surface load. Values associated with this specification are on a new input line with a (4f16.9) format. The command format in Jobname.CDB is:

```
SFE, ELEM, LKEY, Lab, KEY, R5.0
```

- **ELEM**
  The element number.

- **LKEY**
  The load key associated with this surface load.

- **Lab**
  A label indicating the type of surface load: Valid labels are:
  - PRES (pressure)
  - CONV (convection)
  - HFLU (heat flux)
  - IMPD (impedance)
  - SEL (substructure load vector)
  - SELV (S. E. load vectors)
  - CHRG (charge density)

- **KEY**
  A value key. If it is 1, the values are real (film coefficient if convection). If it is 2, values are imaginary (bulk temperature if convection).
Chapter 4: ANSYS Graphics File Format

4.1. Modifying ANSYS Graphics Files

Some ANSYS users may wish to translate ANSYS graphics files to other formats (such as Encapsulated PostScript or AI). If you plan to translate graphics files, this chapter provides some information to help you:

- Source code, with comments, for a program called PLOT33 which plots all of the plots on the coded plot file. You can link this program with user-supplied Calcomp HCBS software (plot, plots, symbol) or other software that uses the Calcomp subroutine protocol. For example, Hewlett-Packard's ISPP product, Tektronix's Preview product, and Veratec's Versaplot product use the Calcomp protocol.

To work with other plotter software, you may need to remove the calls to the Calcomp subroutine (plot, plots, and symbol) and substitute the subroutine calls for the plotter software being used.

Note

Because ANSYS customers use a wide variety of plotters, ANSYS, Inc. can not verify the use of the PLOT33 program with all plotter types.

- A description of the format for the ANSYS neutral graphics file. This listing excludes format information for z-buffered graphics, but the PLOT33 program does include a section on z-buffered graphics.

4.2. Pixmap Format for Graphic Display Files

The ANSYS graphics display is $KPx$ pixels high by $KPx \times 1.33$ pixels wide.

$KPx$ is the resolution specified by the /GFILE,SIZE command (where SIZE is the pixel resolution) or by choosing menu path Utility Menu>PlotCtrls>Redirect Plots>To File. Default resolution is 800.

$IX1,IY1$ is the lower left corner of the z-buffer image.

$IX2,IY2$ is the upper right corner of the z-buffer image.

The image should be mapped to the hardcopy device accordingly.

The following graphic illustrates the items described above:
4.3. Neutral Graphics File Format

The neutral graphics file is an 80-byte, ASCII coded file with fixed length records. It contains plot directives representing the image of a display, as formed in ANSYS, encoded onto a host-independent, printable character set.

Most ANSYS users will not need to know the format of the graphics file. However, in rare cases, you may want to edit your graphics file or, as a programmer, you may need to know the file format to write a program that reads it. Although the file is ASCII coded, it can be difficult to interpret. This section gives details about the file format.

4.3.1. Characters the Graphics File Uses

The host-independent printable character set consists of the ASCII characters listed below:

- Numerals 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9
- Uppercase alphabetic characters A through Z
- The following characters: $ () * + , - . < = >
- The space character, " ".
4.3.2. Graphics File Directives

Graphics files contain a set of directives that define various aspects of how ANSYS displays a plot, such as window coordinates, colors for graphs and text, line dimensions, and so on. Each directive consists of a command character followed by one or more parameters.

Within a graphics file, one directive directly follows the preceding directive. For example, below is the first line of a graphics file:

```
(BAAA2A0AAAAAPPPLPO>AP$MEKLKBAJANSYS 5.3$MEKLEFALNOV 15 1996$MEKKOJAI10:01:40
```

The text of this example line breaks down as follows:

- **(BAAA**
  - The Start-Plot directive, beginning with command character. (B, B, A, A, and A are the values of parameters defining the plot environment. (Parameters for all plot directives, and their possible values, are explained later.)

- **2A**
  - The Text-Size directive, which determines the type size of displayed text strings. The 2 is the command character, and A represents the size value.

- **0AAAAAAPPPLPO**
  - The Window directive, which sets the coordinates for the displayed image. 0 is the command character. AAAAAA represents the first set of coordinates (the lower left corner of the image), and PPPLPO represents the second coordinate set (the right upper corner of the image).

- **>AP**
  - The Text-Color directive, which sets the color of displayed text. > is the command character. AP is a parameter value specifying the color.

- **$MEKLKBAJANSYS 5.3**
  - The first of several Text directives. $ is the command character, MEKLKB are the coordinates for the text, AJ is the number of characters in the string, and ANSYS 5.3 is the text string itself.

- **$MEKLEFALNOV 15 1996**
  - A second Text directive, defining the position and length of the string NOV 15 1996.

- **$MEKKOJAI10:01:40**
  - A third Text directive, defining the position and length of the string 10:01:40

### 4.3.2.1. Parameter Types for Graphics File Directives

The descriptions of graphics file directives in the next section include discussions of the parameter or parameters for each directive. There are five types of parameters:

<table>
<thead>
<tr>
<th>Parameter Type</th>
<th>Parameter Attributes</th>
<th>Valid Parameter Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td>1 byte, base 16 (letters A through P)</td>
<td>0 through 15</td>
</tr>
<tr>
<td>Long</td>
<td>2 bytes, base 16 (letters A through P)</td>
<td>0 through 255</td>
</tr>
<tr>
<td>Byt3</td>
<td>3 bytes, base 16 (letters A through P)</td>
<td>0 through 65535</td>
</tr>
</tbody>
</table>
### 4.3.2.2. Directive Descriptions

The next few pages describe each of the graphics file directives. Parameters are always specified in the order shown below.

<table>
<thead>
<tr>
<th>Graphics Directive</th>
<th>Command Character</th>
<th>Parameters</th>
<th>Parameter Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start_Plot</td>
<td></td>
<td>keras - Defines whether the display surface is cleared prior to the plot (0 = do not clear the surface, 1 = clear it)</td>
<td>Int, Int, Int, Int, Int</td>
</tr>
<tr>
<td></td>
<td></td>
<td>kras - Defines whether the display uses raster mode or vector mode (1 = raster mode, 0 = vector mode)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>kcntr - Defines whether the display uses a contour color map or shading color map (1 = contour, 0 = shading)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>kdocu - Defines whether the Docu column is compressed (1 = do not compress, 0 = compress)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ispare - A spare value</td>
<td></td>
</tr>
<tr>
<td>Window</td>
<td>0</td>
<td>x1,y1,x2,y2 (x and y coordinates)</td>
<td>Xy, Xy</td>
</tr>
<tr>
<td>Area-Color</td>
<td>&lt;</td>
<td>iclra - Sets the color for the displayed area. (See &quot;Color Specification&quot; below.)</td>
<td>Long</td>
</tr>
<tr>
<td>Graph-Color</td>
<td>=</td>
<td>iclrg - Sets the color for the displayed graph. (See &quot;Color Specification&quot; below.)</td>
<td>Long</td>
</tr>
<tr>
<td>Text-Color</td>
<td>&gt;</td>
<td>iclrt - Sets the color for displayed text. (See &quot;Color Specification&quot; below.)</td>
<td>Long</td>
</tr>
<tr>
<td>Text-Size</td>
<td>2</td>
<td>tsise - Defines the size of displayed text (0 = normal, 1 = small)</td>
<td>Int</td>
</tr>
<tr>
<td>Line-Type</td>
<td>,</td>
<td>itype - Defines the type of lines used in the display (0 = solid, 1 = dashed)</td>
<td>Int</td>
</tr>
<tr>
<td>Line-Width</td>
<td>1</td>
<td>ilwidth - Defines the width of displayed lines (0 = normal, 1 to 5 = larger line size)</td>
<td>Int</td>
</tr>
<tr>
<td>Marker Size</td>
<td>3</td>
<td>size - Defines the size of the node marker (0 = the smallest size, 15 = the largest size)</td>
<td>Int</td>
</tr>
<tr>
<td>Point</td>
<td>*</td>
<td>x,y - Defines a point at coordinates x,y</td>
<td>Xy</td>
</tr>
<tr>
<td>Move</td>
<td>.</td>
<td>x,y - Moves to coordinates x,y</td>
<td>Xy</td>
</tr>
<tr>
<td>Draw</td>
<td>-</td>
<td>x,y - Draws a line to coordinates x,y</td>
<td>Xy</td>
</tr>
<tr>
<td>Text</td>
<td>$</td>
<td>x,y - Sets coordinates for where text will display</td>
<td>Xy, Long, String</td>
</tr>
<tr>
<td></td>
<td></td>
<td>nchar - Defines the number of displayed characters</td>
<td></td>
</tr>
</tbody>
</table>
### 4.3.2.3. Color Specification

Below is the list of color specifications used by the directives that set colors for areas, graphs, and text. If more than a single intensity of a color is available, use the value specified by the `Normal` directive to complete the selection. `Normal` of 0 represents the lowest intensity and `normal` of 255 represents the highest intensity.

<table>
<thead>
<tr>
<th>Value</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Black</td>
</tr>
<tr>
<td>1</td>
<td>Cyan</td>
</tr>
<tr>
<td>2</td>
<td>Blue-Magenta</td>
</tr>
<tr>
<td>3</td>
<td>Red</td>
</tr>
<tr>
<td>4</td>
<td>Cyan-Blue</td>
</tr>
<tr>
<td>5</td>
<td>Magenta-Red</td>
</tr>
<tr>
<td>6</td>
<td>Green</td>
</tr>
<tr>
<td>7</td>
<td>Orange</td>
</tr>
<tr>
<td>8</td>
<td>Magenta</td>
</tr>
<tr>
<td>9</td>
<td>Yellow-Green</td>
</tr>
</tbody>
</table>
### Value | Color
--- | ---
10 | Blue
11 | Green-Cyan
12 | Yellow
13 | Dark Gray
14 | Light Gray
15 | White
16 | Reserved for future use
127 | 
128 | Blue
129 | Cyan
130 | Green
Indices 128 through 255 represent the color spectrum used to display the Low (Blue) to High (Red) contour values.
131 | Yellow
132 | Orange
255 | Red

---

### 4.4. Decoding a Graphics File: an Example

The following example shows you the following:

- The ANSYS command stream used to create a simple graphics plot, shown in *Figure 4.2, “Example Display of a Graphics File”* below
- The encoded graphics file that these commands produce
- The decoded graphics plot directives
4.4.2. Example Graphics File Contents

The commands listed above produce the display shown in Figure 4.2, “Example Display of a Graphics File” and the following graphics file:

(BBAAA2A0AAAAAAPPLPO#60AA#62AP#MEKLKBAJANSYS 5.3#MEKLEFALNOV 16 1996#MEK KOJA115:57:07#MEKKIAMMPLLOT NO. 1$MEKKDAIELEMENTS$MEKJNEAIENUMN2 B0AAAAALPOLPO#60AB/PP+EBBHBBHKOGBBHOGKOGBBHKOG$PPFPAB1$AIL1LAB1$L HCA1LAB2LHCLHCAB4$AILLHCAB32A0AAAAAPPLPC.AAAAAA-LPOAAL-POLPO-AAL PO-AAAAA>AB5ABLLKKBAB1>AP#MEKJBLAGZV =1$MEKILPAJDIST=0.55$MEK1GCAIXF =1.5$MEK1AGAYF =0.5$MEKHKAPCENTROID HIDDENAB0A0BOCA ANSYS 5.3 Example Graphics File)

The decoded plot directives are:
Chapter 4: ANSYS Graphics File Format

<table>
<thead>
<tr>
<th>BBAAA</th>
<th>Start-Plot: /ERASE, raster mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>2A</td>
<td>Text-Size: Default</td>
</tr>
<tr>
<td>0AAAAAPPPLPO</td>
<td>Window: 0.0 4095,3070</td>
</tr>
<tr>
<td>6#60AA</td>
<td>Area-Color: Black</td>
</tr>
<tr>
<td>6#62AP</td>
<td>Text-Color: White</td>
</tr>
<tr>
<td>$MEKLBANANSYS 5.3</td>
<td>Text: 3146 2977 &quot;ANSYS 5.3&quot;</td>
</tr>
<tr>
<td>$MEKLEFALNOV 16 1996</td>
<td>Text: 3146 2885 &quot;NOV 15 1996&quot;</td>
</tr>
<tr>
<td>$MEKJOJA15:57:07</td>
<td>Text: 3146 2793 &quot;15:57:07&quot;</td>
</tr>
<tr>
<td>$MEKIMAMPLOT NO. 1</td>
<td>Text: 3146 2700 &quot;PLOT NO. 1&quot;</td>
</tr>
<tr>
<td>$MEKDDAIELEMENTS</td>
<td>Text: 3146 2608 &quot;ELEMENTS&quot;</td>
</tr>
<tr>
<td>$MEKJNEAELELEM NUM</td>
<td>Text: 3146 2516 &quot;ELEM NUM&quot;</td>
</tr>
<tr>
<td>2B</td>
<td>Text-Size: Small</td>
</tr>
<tr>
<td>0AAAAALPOLPO</td>
<td>Window: 0.0 3070 3070</td>
</tr>
<tr>
<td>6#60AB</td>
<td>Area-Color: Cyan</td>
</tr>
<tr>
<td>/PP</td>
<td>Normal: 255</td>
</tr>
<tr>
<td>+EBBBBHHKOBGKHGKOBGKHOG</td>
<td>Polygon: 279,279,2790,2790,2790,2790</td>
</tr>
<tr>
<td>$FFPFPPAB1</td>
<td>Text: 1535 1535 &quot;1&quot;</td>
</tr>
<tr>
<td>$AILAILAB1</td>
<td>Text: 139 139 &quot;1&quot;</td>
</tr>
<tr>
<td>$ILHCAILAB2</td>
<td>Text: 2930 139 &quot;2&quot;</td>
</tr>
<tr>
<td>$ILHCLHAB4</td>
<td>Text: 2930 2930 &quot;4&quot;</td>
</tr>
<tr>
<td>$AILHCBAB3</td>
<td>Text: 139 2930 &quot;3&quot;</td>
</tr>
<tr>
<td>2A</td>
<td>Text-Size: Default</td>
</tr>
<tr>
<td>0AAAAAPPPLPO</td>
<td>Window: 0.0 4095,3070</td>
</tr>
<tr>
<td>.AAAAA</td>
<td>Move: 0,0</td>
</tr>
<tr>
<td>-LPAAAA</td>
<td>Draw: 3070,0</td>
</tr>
<tr>
<td>-LPOLPO</td>
<td>Draw: 3070,3070</td>
</tr>
<tr>
<td>-AAALPO</td>
<td>Draw: 0,3070</td>
</tr>
<tr>
<td>-AAAAA</td>
<td>Draw: 0,0</td>
</tr>
<tr>
<td>&gt;AB</td>
<td>Text-Color: Cyan</td>
</tr>
<tr>
<td>$ABLLLKBAB1</td>
<td>Text: 27 2977 &quot;1&quot;</td>
</tr>
<tr>
<td>&gt;AP</td>
<td>Text Color: White</td>
</tr>
<tr>
<td>$MEKJBLAZGZ =1</td>
<td>Text: 3146 2331 &quot;ZV =1&quot;</td>
</tr>
<tr>
<td>$MEKILPAJDIST=0.55</td>
<td>Text: 3146 2239 &quot;DIST=0.55&quot;</td>
</tr>
<tr>
<td>$MEKIGCAIXF=1.5</td>
<td>Text: 3146 2146 &quot;XF =1.5&quot;</td>
</tr>
<tr>
<td>Line</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>$\text{MKEIAGAIYF} = 0.5$</td>
<td>Text: 3146 2054 &quot;YF = 0.5&quot;</td>
</tr>
<tr>
<td>$\text{MKEHKKAPCENTROID HIDDEN}$</td>
<td>Text: 3146 1962 &quot;CENTROID HIDDEN&quot;</td>
</tr>
<tr>
<td>$\text{ABOABOCA ANSYS 5.3}$</td>
<td>Text: 30 30 &quot;ANSYS 5.3 Example Graphics File&quot;</td>
</tr>
<tr>
<td>)</td>
<td>End-Plot</td>
</tr>
<tr>
<td></td>
<td>No-Op</td>
</tr>
</tbody>
</table>
Part II, Guide to ANSYS User Programmable Features
Chapter 5: Using User Programmable Features (UPFs)

Because the ANSYS program has an open architecture, you can write your own routines or subroutines in C or FORTRAN and either link them to ANSYS or use them as external commands. In fact, some of the ANSYS features you see today as "standard" offerings originated as user programmable features (UPFs). You can take advantage of UPFs if you are licensed for any of the following products:

- ANSYS Multiphysics
- ANSYS Mechanical
- ANSYS Structural
- ANSYS PrepPost
- ANSYS Academic Associate
- ANSYS Academic Research
- ANSYS Academic Teaching Advanced
- ANSYS Academic Teaching Mechanical

Other versions of the ANSYS program do not support UPFs.

5.1. What Are UPFs?

User programmable features are ANSYS capabilities you can use to write your own routines. Using UPFs, you can tailor the ANSYS program to your organization's needs. For instance, you may need to define a new material behavior, a special element, or a modified failure criterion for composites. You can even write your own design optimization algorithm that calls the entire ANSYS program as a subroutine.

UPFs provide the following capabilities:

- To read information into or fetch information from the ANSYS database, you can create subroutines and either link them into the ANSYS program or use them in the external command feature (see Appendix A; Creating External Commands in UNIX for more information about external commands). If you link these subroutines into ANSYS, you are limited to 10 database access commands. Such commands, created through either method, operate at all levels of ANSYS operation, including the begin, preprocessor, general postprocessor, time-history postprocessor, and solution levels. For more information about accessing the ANSYS database, see Chapter 7: Accessing the ANSYS Database.

- ANSYS provides a set of routines you can use to specify various types of loads, including BF or BFE loads, pressures, convections, heat fluxes, and charge densities. These routines are described under Section 6.5: Routines for Customizing Loads.

- Another set of UPF routines enables you to define the following material properties: plasticity, creep, swelling law, viscoplasticity, hyperelasticity, and layered element failure criteria. To see inputs and outputs for these routines, see Section 6.4: Routines for Customizing Material Behavior.

- Several sets of UPFs enable you to define new elements and to adjust the nodal orientation matrix. See Section 6.1: Creating a New Element for more information.

- Another group of UPFs enables you to modify and monitor existing elements. For details, see Section 6.3: Routines for Modifying and Monitoring Existing Elements.
• You can customize UPF userop to create a custom design optimization routine. For more information, see Section 6.10: Creating Your Own Optimization Routine.

• You can call the ANSYS program as a subroutine in a program you have written. To learn how, see Section 6.6: Running ANSYS as a Subroutine.

5.2. What You Should Know Before Using UPFs

Before you do anything with linked UPFs, contact your on-site ANSYS system support person to get the permissions needed to access the appropriate ANSYS files.

The UPF subroutines are written in FORTRAN 90; some extensions are used. They contain comments intended to give you enough detail to develop your own versions of the subroutines.

User routines that can be modified have the word “user” in the first line of the routine. Other routines and functions described in this document that do not have “user” in the first line cannot be modified and must be used as is.

To use UPFs successfully, you need strong working knowledge of the following:

• The ANSYS program.

• The UPF subroutines themselves. Study the UPF subroutines before customizing them, and make sure that you fully understand the subroutines, as well as any applicable functions. Unless you review them carefully, a few UPF subroutines may seem like a maze with many logic paths to consider. You may have to set special variables correctly in order to run your customized ANSYS program without errors. Even if you have in-depth knowledge of the ANSYS input and your desired outputs, you still need to ensure that everything that needs to be done in the UPF subroutines is done properly in your custom version.

• FORTRAN 90. Besides knowing how to write FORTRAN 90 subroutines, you must be sure that the level of the FORTRAN 90 compiler is as least as high as the level mentioned in your ANSYS installation manual. You also need to know what to do should the computer abort the program due to an arithmetic error, a file read error, a memory access error, and so on.

• The mathematics of the phenomenon you are planning to include.

Important

• UPFs are not available or will behave unpredictably in certain data center environments or on some hardware configurations. You should take special care when using UPFs on parallel systems. You should never use the /CONFIG command or a config.ans file to activate parallelization on a system with UPFs. For additional information, consult your ANSYS installation manual or your on-site ANSYS system support person.

• Carefully consider whether you wish to use UPFs, especially if you are linking them into ANSYS (rather than into a shared library for use as external commands). When you add your own routines to ANSYS by either method, you are creating a customized, site-dependent version of the program. ANSYS, Inc. considers the use of UPFs a nonstandard use of the program, one that the ANSYS Quality Assurance verification testing program does not cover. Therefore, you are responsible for verifying that the results produced are accurate and that your customizations do not adversely affect unchanged areas of the ANSYS program.

• Although the flexibility that UPFs offer can be highly attractive, UPF usage is a complicated process that can introduce errors. Consider what you want your customizations to accomplish. You may be able to customize ANSYS more easily and safely with macros than with UPFs.

For other guidelines for nonstandard uses of the ANSYS program, see the Advanced Analysis Techniques Guide.
5.3. Planning Your UPFs

UPFs can range from a simple element output routine for customized output to a complex user optimization. Before you start programming, ask yourself these questions:

- Does the capability you want already exist in the ANSYS program? Remember, a capability may not be obvious at first, especially to a novice ANSYS user.
- Does your proposed subroutine fit into the ANSYS program architecture and specifications? For example, you cannot program a user element that has more than 32 degrees of freedom per node.

Use your experience and judgment to answer these questions. If you need help to do so, consult your ANSYS Support Distributor. If you can respond "no" to both questions, then the user routine you are planning will be both useful and feasible.

5.4. Studying the ANSYS User Routines

Your ANSYS distribution medium contains the source codes for all user routines:

- If you have a UNIX version of ANSYS, the source code for the UPF routines resides in directory /an-
sys_inc/v110/ansys/customize/user/<platform>.
- If you are running the ANSYS program under Windows, the UPF source code resides in directory Program
Files\Ansys Inc\V110\ansys\custom\user\<platform>.

Most of the user routines have at least simple functionality, so print out the routines of interest before you start programming. All source routines are concatenated onto file user.f or user.for. Delete the routines you do not want and make appropriate changes to the others.

5.5. Programming in Languages Other than FORTRAN

If you access UPFs by compiling and linking a custom version of ANSYS, the preferred method is to design and program your custom routine in FORTRAN 90. Although you can use languages other than FORTRAN 90, in each case FORTRAN 90 must provide the interface to the rest of the ANSYS program. If you do use a language other than FORTRAN 90, such as the C programming language, your code may require a FORTRAN shell.

You need to take care when calling FORTRAN subroutines from C subroutines. You must use the symbol associated with the FORTRAN subroutine when invoking the subroutine from a C function. This symbol typically differs slightly from the FORTRAN subroutine name, and is extremely system dependent.

On many UNIX systems, you build this symbol name by taking the FORTRAN subroutine name, converting it to lower case, and appending an underscore. For example, the symbol name for the FORTRAN subroutine HeapInquire would be heapinquire_. You would have to use the symbol heapinquire_ in the invoking C function to avoid an unsatisfied external reference when the program is linked.

Keep in mind that the instance described above is just an example. Compilers from different vendors may construct the symbols differently. Please consult the manuals for your specific compiler for information on how to call FORTRAN subroutines from C functions.

5.6. Developing UPFs: a Suggested Strategy

When developing UPFs by compiling and linking a custom version of ANSYS, you can avoid problems and reduce debugging time by following a gradual, orderly process. Start with a trivial test. Then, add a few changes at a time so that if something goes wrong, the error that caused the problem should be isolated and relatively easy to locate.
The example procedure below illustrates this type of gradual process. The example assumes that you are creating a new element for the ANSYS program using the method described in Section 6.1.2: Creating a New Element by Directly Accessing the ANSYS Database. You develop and test it by performing these steps:

1. Get the applicable element subroutines for \texttt{uel101} from the ANSYS distribution medium. Add a small change (such as a misspelling in an output heading), then compile and link the subroutines.
2. Using a production version of the ANSYS program, run several analysis problems using \texttt{LINK8} (and maybe other elements) to form a base for comparison.
3. Replacing \texttt{LINK8} with \texttt{USER101}, run the same problem on your custom version of ANSYS.
4. Compare the results from Steps 2 and 3. If they show discrepancies other than the misspelled output heading, resolve them before you go on to Step 5.
5. Choose the standard ANSYS element that most closely resembles your new custom element, and run some problems on a production version of ANSYS using that element.
6. Modify the element subroutines to match the element you chose in Step 5. Then, compile and link those subroutines into a custom version of ANSYS.
7. Again, compare the results from Steps 5 and 6. If they don’t match, resolve the discrepancies before moving on to Step 8.
8. Modify your element subroutines to include the features you want. Then, compile and link the subroutines into a custom version of ANSYS.
9. Test the changes with a series of increasingly complex problems for which you already know the answers.

5.7. Include Decks

In addition to the subroutines and functions described in this chapter, most of the include decks (files with the extension \texttt{.inc}) used by ANSYS are on your ANSYS distribution medium. These include decks, also called \textit{commons}, contain important but relatively small amounts of data. The ANSYS program also handles large amounts of data using various access routines (\texttt{GET} and \texttt{PUT}), as described elsewhere in this manual.

To insert include decks in a subroutine or function, use the \texttt{INCLUDE} (or an analogous) statement. \textit{Do not modify an include deck under any circumstances}. The following table lists some of the more commonly used ANSYS include files and the definitions they contain:

<table>
<thead>
<tr>
<th>Include File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acelmcm.inc</td>
<td>Contains accelerations and angular velocities</td>
</tr>
<tr>
<td>ansys-def.inc</td>
<td>Defines general ANSYS parameters. You must include this common to retrieve the parameter values of \texttt{MEM_INTEGER}, \texttt{MEM_DOUBLE}, \texttt{MEM_COMPLEX}, or \texttt{MEM_REAL}.</td>
</tr>
<tr>
<td>cmopt.inc</td>
<td>Contains optimization variables</td>
</tr>
<tr>
<td>echprm.inc</td>
<td>Defines parameters for element characteristics</td>
</tr>
<tr>
<td>elecmt.inc</td>
<td>Defines element characteristics (comments only)</td>
</tr>
<tr>
<td>elecom.inc</td>
<td>Contains element-specific information</td>
</tr>
<tr>
<td>elparm.inc</td>
<td>Defines pointers for the element data array</td>
</tr>
<tr>
<td>elucm.com</td>
<td>Defines the element degree of freedom pointers</td>
</tr>
<tr>
<td>etycm.com</td>
<td>Element type data</td>
</tr>
<tr>
<td>impcom.com</td>
<td>Used by all routines and functions in the ANSYS program</td>
</tr>
<tr>
<td>outpcm.com</td>
<td>Defines output control information</td>
</tr>
<tr>
<td>soptcm.com</td>
<td>Contains solution options and keys</td>
</tr>
</tbody>
</table>
5.8. Linking User Routines

After you make your changes to the user routines supplied on your ANSYS distribution medium, you can either:

- Link your routines into shared libraries (as discussed starting in Appendix A: Creating External Commands in UNIX).
- Compile and link your custom routines into the ANSYS program itself. This is discussed for UNIX systems in Section 5.9: Compiling and Linking UPFs on UNIX Systems and for Windows systems in Section 5.10: Compiling and Linking UPFs on Windows Systems. You may need superuser or root privileges to run the procedure that does the linking.

5.9. Compiling and Linking UPFs on UNIX Systems

As mentioned previously, the source files for the user routines reside in subdirectory `/ansys_inc/v110/ansys/customize/user/<platform>`. If you modify any of these subroutines, select the Relink ANSYS option from ANS_ADMIN utility to link these changes.

When you run a user-linked version of the ANSYS program, the ANSYS output will include the following:

NOTE: This ANSYS version was linked by Licensee

The Relink ANSYS option compiles all FORTRAN files (files ending with `.F`) and all C files (files ending with `.c`) in the current working directory. The procedure then loads all object files (files ending with `.o`) along with the default ANSYS objects and libraries in `/ansys_inc/v110/ansys/customize/user/<platform>` (where `platform` is a directory that uniquely identifies the hardware platform version). The new executable file created will be named `ansycust.e110` and will reside in the current directory.

FORTRAN files are assumed to be FORTRAN 90 (some extensions are allowed), and C files are assumed to be ANSI C.

When relinking on UNIX systems, you can choose to link a distributed version of ANSYS as well. If you choose to link a distributed version, the executable (`ansycustdis.e110`) must reside in the same directory path on all systems. However, you need to link it on only one system; you can then copy the executable to the other systems. You cannot link a distributed version on Windows systems.

The Installation and Configuration Guide lists the compilers you will need to use UPFs.

Creating a Shared Library You can also set up UPFs on some UNIX and Linux systems through a shared library as an alternative to creating a custom ANSYS executable. Copy the ANSUSERSHARED script from `/ansys_inc/v110/ansys/customize/user/<platform>` into your working directory. All Fortran (`*.F`) and C (`*.c`) files that you want to include in your shared library should also reside in your working directory. To compile all `*.F` and `*.c` routines, issue the following command:

```
sh ./ANSUSERSHARED
```

If the compile was successful, you will be asked if a shared file is to be created. Choose Yes to create a shared library named `libansuser.so` (or `.sl`).

---

### Include File | Description
--- | ---
stack.inc | Defines stack storage. You must include this common in any routines that access stack space.
stepecm.inc | Contains load step information
usvrcm.inc | Defines storage of user-defined variables
Chapter 5: Using User Programmable Features (UPFs)

To use this library, set the **ANS_USER_PATH** environment variable to point to the working directory where the libansuser shared library resides. Use one of the following commands, depending on the UNIX shell you are using:

```
setenv ANS_USER_PATH workingdirectory
```

or

```
export ANS_USER_PATH=workingdirectory
```

When you run a user-linked version of the ANSYS program, the ANSYS output will echo the value of **ANS_USER_PATH** and will include the following:

```
NOTE:  This ANSYS version was linked by Licensee
```

To return to the original version of ANSYS, unset the **ANS_USER_PATH** environment variable.

ANSYS recommends using the **ANSUSERSHARED** script as a template to try compilers that are not supported by ANSYS, Inc., such as the GNU compilers. To do so, edit the **ANSUSERSHARED** script, making changes to the appropriate platform logic. Note that if you do use compilers other than those listed in the ANSYS Installation and Configuration Guides, you will need to debug (i.e., find missing libraries, unsatisfied externals, etc.) them yourself. ANSYS, Inc. does not provide assistance for customers using unsupported compilers or if the resulting objects are not compatible with the ANSYS executable(s) as distributed.

**Note**

This shared library method is not available on Windows or IBM platforms.

### 5.10. Compiling and Linking UPFs on Windows Systems

As mentioned previously, the source files for the user routines reside in subdirectory Program Files\Ansys Inc\V110\ansys\custom\user\<platform> (where `<platform>` is a directory that uniquely identifies the hardware platform version.)

**Note**

If you intend to modify any of the user routines, make a duplicate copy of the Program Files\Ansys Inc\V110\ansys\custom\user\<platform> directory to preserve the original files for later use, if necessary.

If you modify any of these subroutines, select the **Relink ANSYS** option from the **ANS_ADMIN** utility to link these changes into the ANSYS program. This procedure compiles all FORTRAN files (files ending with `.F`) and all C files (files ending with `.c`) in the Program Files\Ansys Inc\V110\ansys\custom\user\<platform> directory. The procedure then loads all object files (files ending with `.obj`), along with the default ANSYS objects and libraries. The executable file created will be named `ansys.exe` and will reside in Program Files\Ansys Inc\V110\ansys\custom\user\<platform>.

**Note**

**ANS_ADMIN** does not exist on Windows 64-bit systems. In order to relink on Itanium systems, you must open a command prompt window, set your current directory to Program Files\Ansys Inc\V110\ansys\custom\user\win64, and run ANSCUST.

When relinking on a Windows system, you can choose to link with the small export list (“Do you want to link with the small export list?”).
• Answer “No” to link ANSYS with all symbols exported (which may be required by third party interfaces). The link time will be significantly longer. The ANSYS executable provided on the media is linked using this option, and ANSYS recommends that you select this option.

• Answer “Yes” to link ANSYS with the minimum amount of symbols needed to be exported for ANSYS to run. Select this option for the fastest link time.

**Caution**

When creating a custom ANSYS executable, the executable must be named `ansys.exe`. This requirement is due to shared library usage.

After relinking the ANSYS executable, the program can be executed by either of the following two methods:

1. To execute the relinked version of the ANSYS program:
   • Click **Start>Programs>ANSYS 11.0> ANSYS Product Launcher**
   • In the launcher, select the **Customization/Preferences** tab, then browse to the path which contains the relinked `ansys.exe`. Select other desired options then pick **Run** to execute the customized `ansys.exe`.

2. To execute the relinked `ansys.exe` from a Command Prompt window, use one of the following commands.
   • Interactive:
     
     ```
     ansys110 -custom <path> -p <product variable> -g
     ```
   • Batch:
     
     ```
     ansys110 -custom <path> -b -p <product variable>-j jobname -i <input file> -o <output file>
     ```

   where “path” indicates the full path to the relinked `ansys.exe`.

**Note**

Output from a user-linked version will contain the following statement:

```
This ANSYS version was linked by Licensee
```

**Note**

You will need all the compilers specified in the Installation and Configuration Guide to use these user programmable features. The user programmable features are loaded onto the system only if you perform a custom installation and choose to install the customization tools.

### 5.11. Activating UPFs

The ANSYS program activates many UPFs through a specific user action. This can be through a command option or a user selection. For example, to activate a user element created using the method described in Section 6.1.2: *Creating a New Element by Directly Accessing the ANSYS Database*, all you need to do is select it as one of the element types in a model (using the **ET** command). You then set the element attribute pointer (**TYPE** command), and define elements using the solid modeling or direct generation method.

UPFs that are not activated by the means described above must be activated by either of the following methods:

• Issuing the **USRCAL** command
• Choosing menu path **Main Menu>Preprocessor>Loads>-Load Step Opts>-Other>User Routines** or **Main Menu>Solution>-Load Step Opts>-Other>User Routines**.

To activate or deactivate the routines, issue the command **USRCAL**, *Rnam1*, ..., *Rnam9*, where *Rnam1* and *Rnam9* are the names of specific routines. You can specify up to nine routines with one **USRCAL** command, or you can issue multiple **USRCAL** commands.

Issue the command **USRCAL**,NONE to deactivate all valid user subroutines. To list the status of the routines, issue the command **USRCAL**,STAT.

For a list of the user routines that the **USRCAL** command (or its equivalent menu paths) affects, see the **USRCAL** command description in the **Commands Reference**.

If you do not activate the UPFs in this manner, standard ANSYS logic will be used by default. For instance, when you apply a convection load, standard ANSYS logic is the default even if you have a user convection routine linked in. The user convection routine must be activated by the **USRCAL** command or its menu equivalent.

### 5.12. Running Your Custom Executable

You can run a custom executable from the **Customization/Preferences** tab of the launcher:

Enter the full pathname to the custom executable in the **ANSYS Custom Exe** field. Do not include the **-custom** argument.

When run from the command prompt, if no path is specified after the **-custom** argument, the ansys110 script searches the current working directory for the custom ANSYS executable (ansyscust.e110 by default on UNIX or ansys.exe on Windows). If the custom ANSYS executable resides in a separate directory (or has a name other than ansyscust.e110 on UNIX), you can specify a different path and filename after the **-custom** argument.

**Caution**

If you are running on a Windows system and you create a custom ANSYS executable, the executable must be named ansys.exe. This requirement is due to shared library usage.

On UNIX, you can also run your custom executable via command line.

    ansys110 -custom /pathname/ansyscust.e110

### 5.13. Verifying Your Routines

After compiling and linking your new user routine, test and verify it using whatever procedures you think are adequate. Remember, verifying that your customized version of the ANSYS program works properly is your responsibility.

Make certain that your custom version of the ANSYS program performs correctly for the combinations of elements, analysis types, materials, boundary conditions, and so on, that you plan to use. Confirm that the logic you introduced is correct and does not produce any unwanted side effects.

In testing your custom user routines, you also should verify that the changes you have made do not affect standard, non-customized ANSYS features. To do so, you can compare the results of a set of problems from the **Verification Manual** run on the standard version and on the customized version. Input for these problems is also available on your ANSYS distribution medium.
Always remember: your last step, a series of steps, or even your concept may be wrong. Proceed in clear steps, and verify your work as often as possible. Keep intermediate versions of your modified source code on backup media.

**Note**

If you contact your site's ANSYS system support person or any ANSYS, Inc. representative about the performance of a custom version of ANSYS, always tell him or her explicitly that you are using a user programmable feature. If you feel that an error exists in an unrelated feature of the ANSYS program, demonstrate the suspected error in a non-customized, production version of the program before you report the error to an ANSYS, Inc. representative.

### 5.14. Debugging Commands

To debug errors in your user routines, you can use commands and other features not documented in the Commands Reference. Use these commands only for extremely small models with few solution iterations (otherwise, they will generate an excessive amount of output). `/TRACK` and `/DEBUG` are described in detail below. Two other useful commands are `OUTEQ` and `/NERR`. The command `OUTEQ,ON` can be used to output results from all equilibrium iterations. The command `/NERR,-1` causes errors to be reported as before, but the run continues anyway, normally terminating with either a) system abort or b) incorrect answers. The `/NERR,-1` command is intended for program debugging and may generate erroneous results. You should remove this statement before generating solutions for production use.

#### 5.14.1. Tracking the Path of Program Logic

The `/TRACK` command issues a message when the program logic enters and leaves some of the higher level subroutines. Subroutines `TrackBegin` and `TrackEnd` (see Chapter 8: Subroutines for Users’ Convenience) set up the `/TRACK` command. Then, issue the command using the format below:

```
/TRACK,MonLevel,PrintLevel,SumLevel
```

- **MonLevel**: The level for timing monitoring.
- **PrintLevel**: The level for enter/exit printout.
- **SumLevel**: The level at which the timing sum is output.

Each of these arguments can be any value between 0 and 9 (default is 0).

You can use the `/TRACK` command to identify which section of code is causing the program to abort. For example, to flag up to eight levels of subroutines to determine when the program logic enters and leaves them, you would issue the command `/TRACK,8`.

#### 5.14.2. Debugging Elements and Solutions

The `/DEBUG` command generates debugging at various points in the output. You can specify one of three formats for `/DEBUG`: solution debug format, element debug format, and general debug format.

##### 5.14.2.1. Solution Debug Format

Issue the command using this format:

```
/DEBUG,-1,F1,F2,F3,F4,F5,F6,F7,F8,F9
```
F1
1 (provides basic solution control debugging)

F2
1 (provides transient debugging using Newmark constants)
2 (provides transient debugging using velocities and accelerations)

F3
1 (provides element matrix debugging and prints matrix + load vectors, before going into solve)
2 (provides element matrix debugging with load vectors only, before going into solve)
3 (provides element matrix debugging with matrix diagonals and load vectors, before going into solve)

F4
1 (provides auto time stepping debugging)

F5
1 (provides multifield debugging)

F6
1 (provides arc-length debugging)

F7
1 (provides basic Newton-Raphson debugging)
2 (provides Newton-Raphson debugging and prints out-of-balance forces or incremental displacement or each DOF)
3 (provides Newton-Raphson debugging and prints applied loads and n-r restoring force for each DOF)

F8
1,2 (provides displacement vector debugging with displacement pointers)
2 (provides displacement vector debugging with incremental displacement)
3 (provides displacement vector debugging with contact database)

F9
1 (provides temporary programmer debugging)

5.14.2.2. Element Debug Format

Issue the command using this format:
```
/DEBUG, 3, G1, G2, G3, G4, G5, G6, G7, G8, G9
```

G1
1 (provides basic element pass debugging)

G2
1 (provides element displacement and coordinate debugging)

G3
1 (provides element matrix debugging and prints matrix + load vectors, after the element routines)
2 (provides element matrix debugging with load vectors only, after the element routines)
3 (provides element matrix debugging with matrix diagonals and load vectors, after the element routines)
5.14.2.3. General Debug Format

Issue the command using this format:

```
/DEBUG, H1, H2, , H4, H5, , , H9
```

**H1**

1 (provides file header record information)

2 (provides input line (character))

3 (provides input line (decoded))

**H2**

1 (provides wavefront reordering and element checking debugging)

2 (provides meshing debugging)

**H4**

1 (provides nodal coordinate system transformation debugging)

2 (provides displacement updating debugging)

**H5**

1 (provides pre-element debugging, element characteristics debugging, and element field load debugging)

**H9**

-1 (print the progress of the resume (or save) to isolate location of failure)

-99 (resume only the command log information for subsequent LGWRITE command)

5.15. Other Useful Commands

Two other ANSYS commands, **NSVR** and **/UCMD**, can help you implement UPFs. (Neither command has an equivalent GUI path.) Use the **NSVR** command to define the number of extra variables that need to be saved for user programmable element options, such as user plasticity.
Issue the /UCMD command to make a user routine into a custom command. For more information, see Section 6.7: Defining Your Own Commands.

5.16. Generating Output

You can generate output controlled by the /OUTPUT command by using the FORTRAN write statement. The output unit for this statement is usually called IOTT. IOTT may be defined with the function wrinqr. See the discussion on the function wrinqr in Chapter 8: Subroutines for Users’ Convenience for more details.

5.17. Reading Large Data Files More Rapidly

When files containing ANSYS-related data are large, loading them into the ANSYS program or writing them out to an external file can be a slow process. For example, consider an ANSYS problem file which contains nearly 462,000 lines, 150,000 of which contain nodal data and 97,383 of them containing data for elements. Because many of the lines in this file are in command format, the ANSYS program spends a lot of time reading it.

You can shorten the time ANSYS takes to read such files by including two commands in your programs, UPFs, or macros: EBLOCK and NBLOCK. The NBLOCK command converts nodal data into fixed format data blocks (which ANSYS can read more quickly than commands). The EBLOCK command places element data into a fixed format block, one line per element. These commands also compress displacement constraint data to one line per constraint node. See Chapter 3: Using CDREAD and CDWRITE in the Guide to Interfacing with ANSYS for more information on the use of these commands.
Chapter 6: UPF Routines and Functions

This chapter describes the various routines, functions, and commands that allow you to customize the ANSYS program for your specific purpose. The first portion of each routine or function (consisting of comment lines) is shown in most cases.

User routines that can be modified have the word “user” in the first line of the routine. For example, the first line of the userop routine looks like this:

```
*deck, userop       user
```

User routines that do not have “user” in the first line cannot be modified and must be used as is.

6.1. Creating a New Element

ANSYS offers two tools for creating a user-defined element:

- The user-defined element API
- Direct access to the ANSYS database and files

ANSYS recommends the user-defined element API in most cases. The direct-access method is generally for special-purpose use only, or if you are already using preexisting elements created with this method.

This table highlights the differences between the two methods:

<table>
<thead>
<tr>
<th>Interface</th>
<th>User-defined element API</th>
<th>Accessing ANSYS database and files directly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Offers a simpler interface while preserving much of the underlying user-element capability. An understanding of the database routines and the file structure is rarely necessary to use the interface.</td>
<td>No special interface. If an element capability exists for an ANSYS element, it will exist here (with a few exceptions). The logic necessary for using this interface effectively is more complex.</td>
</tr>
<tr>
<td>Relative level of difficulty</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Expected compatibility between versions</td>
<td>High</td>
<td>Medium</td>
</tr>
<tr>
<td>Element names</td>
<td>USER300</td>
<td>USER100 to USER105</td>
</tr>
<tr>
<td>Demonstration logic included on the ANSYS distribution media</td>
<td>4-node quad and 20-node brick elements</td>
<td>MASS21 and LINK8 elements</td>
</tr>
<tr>
<td>Typical linear material access routine</td>
<td>getMatProp</td>
<td>propev</td>
</tr>
<tr>
<td>New nonlinear material properties</td>
<td>Program in usermat.</td>
<td>No special programming has been set up.</td>
</tr>
<tr>
<td>Existing nonlinear material properties</td>
<td>All ANSYS standard structural materials are accessible via ElemGetMat.</td>
<td>Limited capability. Accessible via plastx, creepx, and swellx.</td>
</tr>
<tr>
<td>Non-structural material properties</td>
<td>No special programming has been implemented.</td>
<td>No special programming has been implemented.</td>
</tr>
<tr>
<td>Number of different element types allowed</td>
<td>Effectively, no limit.</td>
<td>Effectively, no limit.</td>
</tr>
</tbody>
</table>
### 6.1.1. Creating a New Element via the User-Defined Element API

Following is the general process for creating your own element via the user-defined element API.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Specify the element type.</td>
<td>Issue the ET and TYPE commands. The name of the element must be USER300.</td>
</tr>
<tr>
<td>2.</td>
<td>Define your new element according to the specified element type.</td>
<td>Issue the USRELEM command. Specify the element characteristics (such as the number of nodes, number of dimensions, number of real constants etc.).</td>
</tr>
<tr>
<td>3.</td>
<td>Specify nodal DOFs.</td>
<td>Issue the USRDORF command. You can specify a maximum of 10 DOFs per USRDORF command; to define additional DOFs, issue the command again. Each node will have the same DOFs. Although you can specify any valid DOFs, the total number of DOFs for your element cannot exceed 480, and the number of DOFs for each node cannot exceed 32.</td>
</tr>
<tr>
<td>4.</td>
<td>Define real constants.</td>
<td>If needed.</td>
</tr>
</tbody>
</table>
| 5.   | Create finite element models. | Use either of these methods:  
  - **Direct generation** – Create elements directly from nodes, using commands such as E, EGEN, EN, ENGEN, or EMORE. (You can also use the CDREAD command if the .cdb file is available.) This method is the only way to create an element with a topology different from that of any standard ANSYS element. |
Meshing commands -- This method is available only if your element has the same topology as that of a standard ANSYS element and you have specified any standard element shape (USRELEM KeyShape value) except ANYSHAPE.

6. Apply boundary conditions and loads. As needed.

7. Specify solution options. If your element has multi-field DOFs (displacements and temperatures), disable default solution settings (SOLCONTROL, OFF).

8. Perform postprocessing. Postprocessing occurs normally as with any other element. Only total strain (or equivalent quantities such as thermal gradient) and stress (or equivalent quantities such as thermal flux) are saved as regular result quantities. Other variables are saved as nonsummable miscellaneous variables in the results file.

Note

Steps 2 and 3 specify data for the ANSYS user-defined element API. All other steps represent standard ANSYS features.

Recommendations and Restrictions

The following recommendations and restrictions apply to user-defined element USER300:

- Verify that your input data for the USRELEM and USRDOF commands are consistent with the values used in the UserElem.F code. For example, if the number of dimensions (NDIM) specified via the USRELEM command is 2, do not change the number of dimensions specified in the UserElem.F routine from 2. A runtime error or incorrect results can occur if the values do not match.

- ANSYS may activate default solution settings automatically according to the USER300 element's DOFs, but the default solution control settings may not be optimal for your element. If any convergence difficulty occurs, try disabling the default solution settings (SOLCONTROL, OFF).

- The USER300 element does not support ANSYS section (SECxxx) commands. For composite beams and layered shells, you must input element data via real constants and code the UserElem.F routine accordingly.

6.1.1.1. Subroutine UserElem (Writing Your Own Elements)

The UserElem routine provides an interface to ANSYS code above the element level. The routine passes all data needed to create a user-defined element and returns all data and results from the element to update the ANSYS database and files. With this API, you can create virtually any element type without having to access ANSYS database and files directly. Two examples are included in this routine: a 4-node quadrilateral 2-D element, and 20-node brick structural element, both for geometric linear analysis. Key options (KEYOPT settings) switch the elements.

The following table shows the input and output arguments, and their definition and usage. Some argument names (such as those pertaining to element matrices and load vectors) are common to structural analyses; however, you can specify argument values appropriate to analyses in other engineering disciplines.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Input (I) or Output (O)</th>
<th>Definition</th>
<th>Purpose</th>
<th>How Defined</th>
</tr>
</thead>
<tbody>
<tr>
<td>elId</td>
<td>I</td>
<td>Element number</td>
<td>Output information</td>
<td>At FE model creation</td>
</tr>
<tr>
<td>Symbol</td>
<td>Description</td>
<td>Output Information</td>
<td>At FE Model Creation</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>--------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------------</td>
<td>----------------------</td>
<td></td>
</tr>
<tr>
<td>matId</td>
<td>Material number</td>
<td>Call material routines</td>
<td></td>
<td></td>
</tr>
<tr>
<td>keyMtx</td>
<td>Formulation request</td>
<td>Specifying which matrices and load vectors to form</td>
<td>ANSYS code</td>
<td></td>
</tr>
<tr>
<td>lumpm</td>
<td>Mass matrix format:</td>
<td>Specifying how to form the mass matrix</td>
<td>LUMPJM command</td>
<td></td>
</tr>
<tr>
<td>nDim</td>
<td>Number of dimensions</td>
<td>Element coding</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nNodes</td>
<td>Number of element nodes</td>
<td>Element coding</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nodes</td>
<td>Element node list</td>
<td>Output</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nIntPnts</td>
<td>Maximum number of element integration points</td>
<td>Element coding</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nUsrDof</td>
<td>Number of element DOFs</td>
<td>Element coding -- The DOFs are ordered in the way in which they are listed via the USRDUM command for each node and repeated for all nodes All element matrices -- DOF values and load vectors must be arranged in the same way</td>
<td></td>
<td></td>
</tr>
<tr>
<td>kEStress</td>
<td>Element stress state</td>
<td>Element coding</td>
<td></td>
<td></td>
</tr>
<tr>
<td>keyAnsMat</td>
<td>Element formulation key:</td>
<td>Specifying how to create material data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>keySym</td>
<td>Flag for symmetricity of element matrices</td>
<td>Element coding</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nKeyOpt</td>
<td>Maximum number of element key options allowed</td>
<td>Element coding</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KeyOpt</td>
<td>Element key options</td>
<td>Branching the user-element codes to different formulations. (This could be equivalent to 100 x 100 different types of elements.)</td>
<td>ET command</td>
<td></td>
</tr>
</tbody>
</table>

Example: If nKeyOpt = 2, only KEYOPT(1) and KEYOPT(2) are allowed.
<table>
<thead>
<tr>
<th>Variable</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>temper</td>
<td>Nodal temperatures at current time</td>
</tr>
<tr>
<td>temperB</td>
<td>Nodal temperatures at the end of the last substep</td>
</tr>
<tr>
<td>tRef</td>
<td>Reference temperature</td>
</tr>
<tr>
<td>kTherm</td>
<td>Key indicating whether a thermal load exists: 1 = Calculate the thermal load, 0 = No thermal load calculation</td>
</tr>
<tr>
<td>nPress</td>
<td>Number of pressure values</td>
</tr>
<tr>
<td>Press</td>
<td>Pressures at nodes of element facets (available only when key-Shape is specified via the USRELEM command) The pressure vector is ordered in the element with the same topology as that in the standard element library. Refer to that element for details.</td>
</tr>
<tr>
<td>kPress</td>
<td>Key indicating whether a pressure load exists: 1 = Calculate the pressure load, 0 = No pressure load calculation</td>
</tr>
<tr>
<td>nReal</td>
<td>Number of real constants</td>
</tr>
<tr>
<td>RealConst</td>
<td>The list of real constants</td>
</tr>
<tr>
<td>nSaveVars</td>
<td>The number of variables saved in the .esav file for the element</td>
</tr>
<tr>
<td>saveVars</td>
<td>The data saved in the .esav file ANSYS saves the data after exiting the UserElem routine and retrieves it immediately before entering UserElem again. It should include kinematic related variables only when the ANSYS material routine is called; otherwise, it should include both kinematic and</td>
</tr>
</tbody>
</table>

Temperature dependence and thermal loads

**BF and BFE commands (if key-Shape is specified in the UserElem routine)**

Temperature dependence and thermal loads

ANSYS code

TREF command

Element coding

USRELEM command

SF and SFE commands

ANSYS code

ANSYS code

### 6.1.1. Creating a New Element via the User-Defined Element API

<table>
<thead>
<tr>
<th>I/O</th>
<th>The data saved in the .esav file</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/O</td>
<td>ANSYS saves the data after exiting the UserElem routine and retrieves it immediately before entering UserElem again. It should include kinematic related variables only when the ANSYS material routine is called; otherwise, it should include both kinematic and</td>
</tr>
</tbody>
</table>

**UserElem routine**
material data. History dependent variables can only be saved/updated when the substep is converged (\texttt{keyHistUpd} = 1).

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
<th>Element Coding</th>
<th>ANSYS Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>xRef</td>
<td>I</td>
<td>Initial coordinates of the element nodes</td>
<td>Element coding</td>
<td>At FE model creation</td>
</tr>
<tr>
<td>xCur</td>
<td>I</td>
<td>Current (deformed) coordinates of element nodes</td>
<td>Element coding</td>
<td>ANSYS code</td>
</tr>
<tr>
<td>TotValDofs</td>
<td>I</td>
<td>Total values of DOFs (displacements for structural analysis)</td>
<td>Element coding</td>
<td>ANSYS code</td>
</tr>
<tr>
<td>IncValDofs</td>
<td>I</td>
<td>Increment values of DOFs occurring at the current substeps</td>
<td>Element coding</td>
<td>ANSYS code</td>
</tr>
<tr>
<td>ItrValDofs</td>
<td>I</td>
<td>Iteration values of DOFs occurring at the last iteration</td>
<td>Element coding</td>
<td>ANSYS code</td>
</tr>
<tr>
<td>VelValDofs</td>
<td>I</td>
<td>First time derivatives of DOFs</td>
<td>Velocities</td>
<td>ANSYS code</td>
</tr>
<tr>
<td>AccValDofs</td>
<td>I</td>
<td>Second time derivatives of DOFs</td>
<td>Accelerations</td>
<td>ANSYS code</td>
</tr>
<tr>
<td>kfstps</td>
<td>I</td>
<td>Key indicating first time entering the element routine: 1 = First time 0 = Other than first time</td>
<td>Initializing data</td>
<td>ANSYS code</td>
</tr>
<tr>
<td>nlgeom</td>
<td>I</td>
<td>Flag indicating whether large displacement/deformation is in effect</td>
<td>Element coding</td>
<td>\texttt{NLGEOM} command</td>
</tr>
<tr>
<td>nrkey</td>
<td>I</td>
<td>Newton-Raphson algorithm key: 1 -- Any nonlinear analysis 0 -- Pure linear analysis</td>
<td>Output</td>
<td>---</td>
</tr>
<tr>
<td>outkey</td>
<td>I</td>
<td>Key indicating output result type: 1 -- This is an output call, the substep is converged, and you can print/save element results 0 -- All other cases</td>
<td>Element coding</td>
<td>ANSYS code</td>
</tr>
<tr>
<td>elPrint</td>
<td>I</td>
<td>Key indicating whether any element output should appear in the print file: 0 = No 1 = Yes</td>
<td>Element coding</td>
<td>\texttt{OUTPR} command</td>
</tr>
<tr>
<td>iott</td>
<td>I</td>
<td>Output file number</td>
<td>The FORTRAN output file number. All information written in the specified</td>
<td>ANSYS code</td>
</tr>
<tr>
<td>key</td>
<td>Description</td>
<td>Value</td>
<td>Notes</td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>-------------</td>
<td>-------</td>
<td>-------</td>
<td></td>
</tr>
<tr>
<td>keyHisUpd</td>
<td>Key to update history-dependent variables: 1 = The substep converged; ready to update history-dependent variables (such as equivalent plastic strain) 0 = Solution not yet converged; cannot update history-dependent variables</td>
<td>I</td>
<td>Element coding</td>
<td></td>
</tr>
</tbody>
</table>

**The following variables are for debug, timing, and convergence-control purposes only. You can usually ignore them.**

<table>
<thead>
<tr>
<th>ldstep</th>
<th>Current load step number</th>
<th>Output Debug</th>
<th>ANSYS code</th>
</tr>
</thead>
<tbody>
<tr>
<td>isubst</td>
<td>Current substep number</td>
<td>Output</td>
<td>ANSYS code</td>
</tr>
<tr>
<td>ieqitr</td>
<td>Current iteration number</td>
<td>Output</td>
<td>ANSYS code</td>
</tr>
<tr>
<td>timval</td>
<td>Current time</td>
<td>Output</td>
<td>ANSYS code</td>
</tr>
<tr>
<td>keyEleErr</td>
<td>Formulation error key: 0 = No error (preset value) 1 = Error occurred in element formulation, possibly due to excessive deformation. (ANSYS will lessen deformation if possible by cutback.)</td>
<td>I/O</td>
<td>Element coding</td>
</tr>
<tr>
<td>keyEleCnv</td>
<td>Element convergence key: 1 = Converged (preset value before calling) 0 = Not converged</td>
<td>I/O</td>
<td>Provides manual control of convergence when you introduce any constraint at the element level (such as volumetric constraint for mixed u-P)</td>
</tr>
</tbody>
</table>

**End of special-purpose variable group**

<table>
<thead>
<tr>
<th>eStiff</th>
<th>Small-deformation stiffness matrix In global Cartesian coordinate system</th>
<th>Solution</th>
<th>Requested when keyMtx(1) = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>eMass</td>
<td>Mass matrix In global Cartesian coordinate system</td>
<td>Solution</td>
<td>Requested when keyMtx(2) = 1</td>
</tr>
<tr>
<td>eDamp</td>
<td>Damping matrix In global Cartesian coordinate system</td>
<td>Solution</td>
<td>Requested when keyMtx(3) = 1</td>
</tr>
<tr>
<td>eSStiff</td>
<td>Stress stiffness matrix In global Cartesian coordinate system</td>
<td>Solution</td>
<td>Requested when keyMtx(4) = 1</td>
</tr>
<tr>
<td>fExt</td>
<td>External load vector In global Cartesian coordinate system</td>
<td>Solution</td>
<td>Requested when keyMtx(5) = 1</td>
</tr>
<tr>
<td>fInt</td>
<td>Internal nodal force vector In global Cartesian coordinate system</td>
<td>Solution</td>
<td>Requested when keyMtx(6) = 1</td>
</tr>
</tbody>
</table>
### UserElem Routine

**elVol**  O  Element volume  |  Output  |  UserElem routine

**elMass**  O  Element mass  |  Output  |  UserElem routine

**elCG**  O  Element centroid coordinates in global Cartesian coordinate system  |  Postprocessing  |  UserElem routine

**nRsltBsc**  I  Number of basic result data saved in result file  |  Specifying the size of RsltBsc  |  ANSYS code

**RsltBsc**  O  Basic result data saved in ANSYS result file  
This variable is accessible via the PRESOL and PRNSOL commands in the standard way and can also be plotted if you specify a Key-Shape value via the USRELEM command.  |  Postprocessing  |  UserElem routine

**nRsltVar**  I  The number of result data to be saved in the result file as non-summable miscellaneous variables  |  Specifying the size of RsltVar  |  USRELEM command

**RsltVar**  O  The result data saved in the result file as non-summable miscellaneous variables  
The data is accessible via the PLESOL command only, but only one value for an element each time  |  Postprocessing  |  UserElem routine

**nElEng**  I  Number of energies Fixed at 3  |  Solution  |  UserElem routine

**elEnergy**  O  Element energy vector:  
elEnergy(1) -- Strain energy  
elEnergy(2) -- Plastic energy  
elEnergy(3) -- Creep energy  |  Output  |  UserElem routine

---

```c
*deck,UserElem
subroutine UserElem (elId, matId, keyMtx, lumpm, nDim, nNodes,
  & Nodes, nIntPnts, nUsrDof, kEStress,
  & keyAnsMat, keySym, nKeyOpt, KeyOpt,
  & temper, temperB, tRef, kTherm,
  & nPress, Press, kPress, nReal, RealConst,
  & nSaveVars, saveVars, xRef, xCur,
  & TotValDofs, IncValDofs, ItrValDofs,
  & VelValDofs, AccValDofs,
  & kfsfps, nlgeom, nrkey, outkey, ePrint, iott,
  & keyHisUpd, ldstep, isubst, ieqitr, timval,
  & keyEleErr, keyEleCnv,
  & eStiff, eMass, eDamp, eSStiff,
  & fExt, fint, elVol, elMass, elCG,
  & nRsltBsc, RsltBsc, nRsltVar, RsltVar,
  & nElEng, elEnergy)

c*************************************************************************
c
```

---

**Programmer’s Manual for ANSYS**, ANSYS Release 11.0, 002328. © SASIP, Inc. All rights reserved. Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.
c *** Notice - This file contains ANSYS Confidential information ***
c
PROGRAMMER SHOULD NOT CHANGE ANY PURE INPUT ARGUMENTS (marked by ....,in)!
c
c elId      (int,sc,in)        element number
c matId     (int,sc,in)        material number of this element
c keyMtx    (int,ar(10),in)    matrix and load vector form requests
  0 = not requested, 1 = requested
  see below for more details
c lumpm    (int,sc,in)        mass matrix format
  = 0 no lumped mass matrix
  = 1 lumped mass matrix
c nDim      (int,sc,in)        number of dimensions of the problem
  (defined on USRELEM command as NDIM)
  = 2 2D
  = 3 3D
c nNodes    (int,sc,in)        number of nodes of the element
  (defined on USRELEM command as NNODES)
c Nodes     (int,ar(nNodes),in) node list of this element
  (defined on USRELEM command as NINTPNTS)
c nIntPnts  (int,sc,in)        maximum number of integration points
  (defined on USRELEM command as NINTPNTS)
c nUsrDof   (int,sc,in)        number of DOFs of this element (matrix and
  load vector size)
c kEStress  (int,sc,in)        kEStress
  (defined on USRELEM command as KESTRESS)
c keyAnsMat (int,sc,in)        key to indicate if ANSYS material
  routine is going to be called
  (defined on USRELEM command as KEYANSMAT)
  = 0, No
  = 1, Yes
c keySym    (int,sc,in)        key to indicate if element matrices
  is symmetric
  (defined on USRELEM command as KEYSYM)
  = 0, symmetric
  = 1, unsymmetric
c nKeyOpt   (int,sc,in)        number of element key options able to be
  used in this routine
  values of element key option defined
  by et or keyopt command for the
  user elements, only the first
  nKeyOpt values are passed in and can
  be used to branch the routine for
  different formulations
c temper    (dp,ar(nNodes),in) nodal temperatures at current time
c temperB   (dp,ar(nNodes),in) nodal temperatures at the beginning of this
  incremental step (substep)
c tRef      (dp,sc,in)         reference temperature
  input: flag for thermal loading
  = 1, Temperatures at nodes are different
  from the reference temperature,
  thermal loading might be needed.
  = 0, Temperatures at nodes are the same
  as the reference temperature,
  thermal loading is not needed.
c kTherm    (int,sc,in)        input: flag for thermal strains
  output: flag for thermal strains
  number of pressure values for this element
  applied elemental face load (pressure)
c kPress    (int,sc,in)        flag for pressure loading
  = 1, pressure load is applied and
  equivalent nodal forces should be
  calculated
  = 0, no pressure loading
nc nReal    (int,sc,in)        number of real constants
RealConst  (dp,ar(nReal),in) user defined real constants
nc nSaveVars (int,sc,in)       number of saved variables
saveVars  (dp,ar(nSaveVars),inout) user saved variables
nc xRef    (dp,ar(nDim,nNodes),in) nodal coordinates in initial configuration
c xCur    (dp,ar(nDim,nNodes),in) nodal coordinates in current configuration
TotValDofs (dp, ar(nUsrDof), in) total values of DOFs (displacements) from time = 0
IncValDofs (dp, ar(nUsrDof), in) incremental values of DOFs (displacements) for the current step
ItrValDofs (dp, ar(nUsrDof), in) iterative values of DOFs (displacements) for the current iteration (normally needed for debug only)
VelValDofs (dp, ar(nUsrDof), in) first time derivatives of DOFs (velocities) (normally not needed)
AccValDofs (dp, ar(nUsrDof), in) second time derivatives of DOFs (accelerations) (normally not needed)
kfstps (int, sc, in) key for the first iteration of first substep of the first load step
    = 1 yes
    = 0 no
nlgeom (int, sc, in) large deformation key [from nlgeom command]
    = 0 NLGEOM, OFF
    = 1 NLGEOM, ON
nrkey (int, sc, in) key to indicate a newton-raphson (incremental) procedure
    = 0 No
    = 1 Yes
outkey (int, sc, in) key to indicate if any element output is to be placed on the print file or the result file
    = 0 No
    = 1 Yes
elPrint (int, sc, in) key to indicate if any element output is to be placed on the print file
    = 0 No
    = 1 Yes
ioitt (int, sc, in) print output file unit number
keyHisUpd (int, sc, in) key to indicate if history-dependent variables need to be updated, like equivalent plastic strain, back stress etc. since the iteration is already converged
    = 0 not converged, don't need to update history dependent variables
    = 1 yes, converged, need to update history dependent variables
ldstep (int, sc, in) current load step number
isubst (int, sc, in) current substep number
ieqitr (int, sc, in) current equilibrium iteration number
timval (int, sc, in) current time value
keyEleErr (int, sc, inout) key to indicate if there is any formulation error, like negative Jacobian. The error could be caused by too large incremental step, illegal model.
    = 0 no error (preset value before calling)
    = 1 some error happens. ANSYS will decide to stop the analysis or cutback the substep (bi-section) based on other user input and information at higher level.
keyEleConv (int, sc, inout) key to flag if this element satisfies the user defined element convergence criterion.
    = 1, yes, the criterion is satisfied or don't have any criterion at all it is preset value before calling
    = 0, no, the element doesn't satisfy element convergence criterion. If this is the case, the iteration will not converge even when both force and displacement converge
--- end of 7 variable group -----
6.1.1.2. Subroutine ElemGetMat (Calling the ANSYS Standard Structural Material Library)

The `ElemGetMat` routine is the API to the ANSYS materials. When you issue the `USRELEM` command after setting the command’s `KEYANSMAT` argument, the routine accesses the ANSYS standard structural material library. It allows you to focus on the kinematic portion of element formulation while ANSYS handles the material part of the formulation. When calling the routine, input the associated material data via the standard method. There is no need to access this routine, only to call it.

The following table shows the input and output arguments, and their definition and usage.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Input(I) or Output(O)</th>
<th>Definition</th>
<th>Purpose</th>
<th>How Defined</th>
</tr>
</thead>
<tbody>
<tr>
<td>elId</td>
<td>I</td>
<td>Element number</td>
<td>Output</td>
<td>At FE model creation</td>
</tr>
<tr>
<td>matId</td>
<td>I</td>
<td>Material number</td>
<td>Output information</td>
<td>At FE model creation</td>
</tr>
<tr>
<td>nDim</td>
<td>I</td>
<td>Number of dimensions of element geometry</td>
<td>Material calculation</td>
<td>At FE model creation</td>
</tr>
</tbody>
</table>

1 = 2-D element geometry
2 = 3-D element geometry
<table>
<thead>
<tr>
<th>Function</th>
<th>Declaration</th>
<th>Description</th>
<th>Data Type</th>
<th>Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>nTens</td>
<td>I</td>
<td>Number of stress/strain tensor components: 4 = 2-D and ordered as x, y, z, xy 6 = 3-D and ordered as x, y, z, xy, yz, xz</td>
<td>Specifying the data size</td>
<td>UserElem routine</td>
</tr>
<tr>
<td>nDirect</td>
<td>I</td>
<td>Number of direct component of stress/strain tensors nDirect &lt; or = nTens</td>
<td>Specifying the data size</td>
<td>UserElem routine</td>
</tr>
<tr>
<td>intPnt</td>
<td>I</td>
<td>Current integration point number</td>
<td>Output</td>
<td>UserElem routine</td>
</tr>
<tr>
<td>xCurIP</td>
<td>I</td>
<td>Coordinates of current integration point Values in global Cartesian coordinate system</td>
<td>Material calculation</td>
<td>UserElem routine</td>
</tr>
<tr>
<td>TemperIP</td>
<td>I</td>
<td>Integration point temperatures at the current time</td>
<td>Evaluating temperature-dependent material data</td>
<td>UserElem routine</td>
</tr>
<tr>
<td>TemperIPB</td>
<td>I</td>
<td>Integration point temperatures at the end of the last incremental step</td>
<td>Evaluating temperature-dependent material data</td>
<td>UserElem routine</td>
</tr>
<tr>
<td>IncStrain</td>
<td>I</td>
<td>Strain components [1] Incremental strain of the current substep when nlgeom = on Total strain when nlgeom = off</td>
<td>Material calculation</td>
<td>UserElem routine</td>
</tr>
<tr>
<td>defG0</td>
<td>I</td>
<td>Deformation gradient tensor at the end of previous substep [1]</td>
<td>Material updating</td>
<td>UserElem routine</td>
</tr>
<tr>
<td>defG</td>
<td>I/O</td>
<td>Total deformation gradient tensor at the current time [1]</td>
<td>The component in thickness direction is updated by material routines for plane stress and shell elements</td>
<td>UserElem routine</td>
</tr>
<tr>
<td>kTherm</td>
<td>I/O</td>
<td>Thermal loading key: 0 = No thermal loading 1 = Has thermal loading</td>
<td>Thermal load calculation</td>
<td>UserElem routine</td>
</tr>
<tr>
<td>cMat</td>
<td>O</td>
<td>Material Jacobian [1]</td>
<td>Forming stiffness</td>
<td>Material routine</td>
</tr>
<tr>
<td>MatProp</td>
<td>O</td>
<td>Material data for element formulation</td>
<td>Forming mass matrix Handling transverse shear Output</td>
<td>Material routine</td>
</tr>
<tr>
<td>Stress</td>
<td>O</td>
<td>Cauchy stress [1]</td>
<td>Forming geometric stiffness Calculating internal forces</td>
<td>Material routine</td>
</tr>
<tr>
<td>Strain</td>
<td>O</td>
<td>Total strain components [1]</td>
<td>Output</td>
<td>Material routine</td>
</tr>
<tr>
<td>StressTh</td>
<td>O</td>
<td>Total thermal stress components [1]</td>
<td>Output</td>
<td>Material routine</td>
</tr>
<tr>
<td>StrainTh</td>
<td>O</td>
<td>Total thermal strain components [1]</td>
<td>Output</td>
<td>Material routine</td>
</tr>
<tr>
<td></td>
<td>Type</td>
<td>Description</td>
<td>Unit</td>
<td>Source</td>
</tr>
<tr>
<td>----------------</td>
<td>------</td>
<td>-------------------------------------------------------</td>
<td>---------------</td>
<td>--------------------</td>
</tr>
<tr>
<td><strong>StrainPl</strong></td>
<td>O</td>
<td>Total plastic strain components [1]</td>
<td>Output</td>
<td>---</td>
</tr>
<tr>
<td><strong>StrainCr</strong></td>
<td>O</td>
<td>Total creep strain components [1]</td>
<td>Output</td>
<td>---</td>
</tr>
<tr>
<td><strong>StressBk</strong></td>
<td>O</td>
<td>Back stress components [1]</td>
<td>Output</td>
<td>---</td>
</tr>
<tr>
<td><strong>StrainSw</strong></td>
<td>O</td>
<td>Swelling strain</td>
<td>Not yet supported</td>
<td>---</td>
</tr>
<tr>
<td><strong>EnergyD</strong></td>
<td>O</td>
<td>Energy density:</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Elastic energy density</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 = Plastic energy density</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 = Creep energy density</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MatRotGlb</strong></td>
<td>O</td>
<td>Rotation matrix from global Cartesian to rotated element coordinate system</td>
<td>Used only for solid elements when nlgeom = on</td>
<td></td>
</tr>
</tbody>
</table>

1. All tensor component values in the routine are in the global Cartesian coordinate system for solid elements, and in the co-rotated element Cartesian coordinate system for link, beam and shell elements.

```
*deck,ElemGetMat

subroutine ElemGetMat (elId, matId, nDim, nTens, nDirect,
 & intPnt, xCurIP, TemperIP,
 & TemperIPB, kTherm, IncStrain,
 & defG0, defG, CMat, MatProp,
 & Stress, Strain, StressTh, StrainTh,
 & StrainPl, StrainCr, StressBk, StrainSw,
 & EnergyD, MatRotGlb)

c*************************************************************************
```

6.1.1. Creating a New Element via the User-Defined Element API
6.1.2. Creating a New Element by Directly Accessing the ANSYS Database

The next few pages describe the user routines and supporting subroutines you use to create new elements. Using these routines, you can create new element types, add them to the ANSYS element library, and use them as "regular" elements. You can create up to six independent element types (names USER100 - USER105). For demonstration purposes, example copies of the routines for MASS21, the structural mass element, and LINK8, the 3-D spar element, are included on the ANSYS distribution medium as uel100 and uel101 respectively.

6.1.2.1. Input and Output Abbreviations

The descriptions of the routines or functions within this chapter describe both the input arguments and output arguments. Argument information includes the argument's type, size and intent.

- **Argument type** is one of the following:
  - int - integer
  - dp - double precision
  - log - logical
  - chr - character
  - dcp - double precision complex

- **Argument size** is one of the following:
  - sc - scalar variable
  - ar(n) - array variable of length n
  - func - functional return value

- **Argument intent** is one of the following:
  - in - input argument
  - out - output argument
  - inout - both an input and an output argument
6.1.2. User Routines

Routines uec100 through uec105 describe the element characteristics. Routine elccmt (on the distribution medium) describes the input for these routines in detail. You can use subroutines uex100 through uex105 to override default logic. Routines uec100 through uec105 define parameters such as:

- 2-D or 3-D geometry
- Degree of freedom set
- Symmetric or unsymmetric matrix
- Number of nodes
- Number of body loads (for example, temperatures)
- Number of surface loads (for example, pressures)
- Number of real constants
- Number of variables to be saved
- Number of rows in element matrices
- Linear or nonlinear element.

Routines uel100 through uel105 calculate the element matrices (stiffness, specific heat, and so on), the element load vector (force, heat flow, and so on), and any element output quantities. The element printout also is generated, and the variables to be saved are calculated and stored in the results file.

Other user routines available for manipulating element information include the following:

- Routines uep100 through uep105 provide printed output of line elements. The general ANSYS postprocessor, POST1, calls the subroutines, or you can call them using uel100 through uel105.
- Routine usertr allows access to the nodal transformations.
- Routine userac describes some of the data handling.

6.1.2.3. Subroutine uec100 (Defining Characteristics of the usr100 Routine)

```verbatim
*deck, uec100 user
  subroutine uec100 (elcdn, ielc, kerr)
  c ***** this subroutine defines the characteristics of user100.
  c
  c *** copyright(c) 2006 SAS IP, Inc. All rights reserved.
  c *** ansys, inc.
  c *** Notice - This file contains ANSYS Confidential information ***
  c
typ=int, dp, log, chr siz=sc, ar(n) intent=in, out, inout
  c
  c input arguments:
  c variable (typ, siz, intent) description
  c ielc (int, ar(IELCSZ), inout) - element characteristics
  c
  c kerr (int, sc, inout) - error flag up to this point.
  c (do not initialize to zero)
  c
  c output arguments:
  c variable (typ, siz, intent) description
  c elcdn (chr, sc, out) - name of element
  c ielc (int, ar(IELCSZ), inout) - element characteristics
  c
  c kerr (int, sc, inout) - error flag (set to 1 if error)
  c note to programmers: the validity of keyopt values may be checked here
```

Programmer's Manual for ANSYS. ANSYS Release 11.0. ©SAS IP, Inc. All rights reserved. Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.
6.1.2.3.1. Subroutines uec101 through uec105

The input and output arguments for subroutines uec101, uec102, uec103, uec104, and uec105 are identical to the uec100 subroutine listed above.

6.1.2.4. Subroutine uex100 (Overriding Element Characteristic Defaults)

```fortran
*deck,uex100                              user
subroutine uex100 (ielc,kerr)
c     *** subroutine to override element characteristic defaults ***
c     *** hence, this routine is needed only in rare cases.
c     *** copyright(c) 2006 SAS IP, Inc.  All rights reserved.
c     *** ansys, inc.
c *** Notice - This file contains ANSYS Confidential information ***
c     *** input and output are the same as for uec100, except that this
c     *** logic is called after the defaulting logic is finished.
c     *** this defaulting is done in ansys subroutine echdft(not a upf).
c     *** as indicated above, this routine is rarely needed, but if it is
c     *** desired to see the effect of echdft, you may print out the ielc array
     *** leaving uec100 and print it out again entering this routine.
c     typ=int,dp,log,chr   siz=sc,ar(n)   intent=in,out,inout

input arguments:
variable (typ,siz,intent)     description
ielc (int,ar(IELCSZ),inout) - element characteristics
kerr   (int,sc,inout)       - error flag up to this point.
                             (do not initialize to zero)
output arguments:
variable (typ,siz,intent)     description
ielc (int,ar(IELCSZ),inout) - element characteristics
kerr   (int,sc,inout)       - error flag (set to 1 if error)
*** standard defaults are taken. the final results are given with
*** the debug accessed with /debug,,, ,,1
```

6.1.2.4.1. Subroutines uex101 through uex105

The source code for subroutines uex101, uex102, uex103, uex104, and uex105 is identical to the code for subroutine uex100 listed above.

6.1.2.5. Subroutine uel100 (Computing Element Matrices, Load Vectors, and Results)

```fortran
*deck,uel100                              user
subroutine uel100 (elem,ielc,elmdat,eomask,nodes,locsvrL,kelreq,
x kelfil,nr,xyz,u,kelout,zs,zaass,damp,gstif,zsc,zscnr,elvol,elmass,
x center,elenes,edindxL,lcerstL)
c --- general lumped mass is demonstrated --------------------------------
c *** primary function:
c 1. compute element matrices, load vectors, and results
 *** secondary functions:
c 2. maintain element solution data

 *** user programmable functions may not be used in parallel processing ***
c *** Notice - This file contains ANSYS Confidential information ***
c *** copyright(c) 2006 SAS IP, Inc.  All rights reserved.
c *** ansys, inc.

input arguments:
```

Chapter 6: UPF Routines and Functions
6.1.2. Creating a New Element by Directly Accessing the ANSYS Database

6.1.2.5.1. Subroutines uel101 through uel105

The input and output arguments for subroutines uel101, uel102, uel103, uel104, and uel105 is identical to subroutine uel100 listed above.

6.1.2.6. Subroutine uep100 (Printing Output for User Elements in POST1 via PRESOL,ELEM)

*deck, uep100    user
subroutine uep100 (iott, elem, nodes, mat, kept, tem,
  x kemn, fluen, kems, force, kens, sig, keel, epel,
  x keth, eptho, epswel, epino, keni, sigepl, sigrat, hpres, epeq,
  x kepl, eppl, kecr, epcr)
  c
  c *** primary function: produce printed output for user100
  c
  c *** copyright(c) 2006 SAS IP, Inc. All rights reserved.
  c
  c *** ansys, inc.
  c
  c ********** this subroutine is provided for user information **********
  c
  c *** user programmable features may not be used in parallel processing ***
  c
  c
  input arguments:
  c   iott (int,sc,in)   - output unit number
  c   elem (int,sc,in)   - element number
  c   nodes (int,ar(2),in) - node numbers
  c   mat (int,sc,in)   - material number
  c   kept (int,sc,in)  - key to print temperatures
6.1.2.6.1. Subroutines uep101 through uep105

The source code for subroutines uep101, uep102, uep103, uep104, and uep105 is identical to subroutine uep100 listed above.

6.1.2.7. Subroutine usertr (Adjusting the Nodal Orientation Matrix)

*deck,usertr                              user

subroutine usertr (node,tr)
  c *** primary function: adjust nodal orientation matrix
  c secondary function: study nodal orientation matrix
  c accessed with ielc(notran) = -100
  c
  c *** copyright(c) 2006 SAS IP, Inc. All rights reserved.
  c *** ansys, inc.
  c
  c *** Notice - This file contains ANSYS Confidential information ***
  c
typ=int,dp,log,chr,dcp   siz=sc,ar(n)   intent=in,out,inout
  c
  c input arguments:
  c variable (typ,siz,intent)    description
  c node     (int,sc,in)       - node number being acted upon
  c tr       (dp,ar(32,32),inout) - nodal to global orientation matrix
  c
  c output arguments:
  c variable (typ,siz,intent)    description
  c tr       (dp,ar(32,32),inout) - nodal to global orientation matrix
  c
  c tr is a matrix that is already defined based on the degrees
  c of freedom selected.
  c it does not normally need to be changed.
  c it may be printed out here to study. its functional size is
  c nr by nr, where nr is the number of degrees of freedom in the
  c element
  c

6.1.2.8. Subroutine userac (Accessing Element Information)

This subroutine is provided for demonstration purposes.
6.2.2. Subroutine svgidx (Fetching the Index for Saved Variables)

*deck, svgidx
  subroutine svgidx (locsvr, svindx)
  c *** primary function: get the index for saved variables
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c   locsvr (LONGINT,sc,in) - pointer to location of index
  c output arguments:
  c   svindx (int,ar(20),out) - the 20 word index of svr variables
c
6.2.2. Subroutine nminfo (Returning Element Reference Names)

*deck, nminfo
  subroutine nminfo (ielc, rname)
  c *** primary function: set element reference names
  c *** secondary functions: none
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c   ielc (int,ar(*),inout) - element characteristic vector
  c   rname (chr,sc,in) - 8 character reference name
  c output arguments:
  c   ielc (int,ar(*),inout) - element characteristic vector with element name encoded

6.2.3. Subroutine svrget (Fetching Saved Variable Data for an Element)

*deck,svrget
subroutine svrget (svindx,nset,nsvr,svr)
c *** primary function: get svr data set for an element
c *** Notice - This file contains ANSYS Confidential information ***
c input arguments:
c svindx (int,ar(20),in) - index for svr for this element (see svgidx)
c nset (int,sc,in) - the set number in this index
  - 1 - structural svrs
  - 2 - thermal/electric/fluid svrs
  - 3 - magnetic svrs
  - 4 - nonlinear svrs
  - 5 - plasticity svrs
  - 6 - creep svrs
  - 7 - coupled svrs
  - 8 - user svrs
  - 9 - initial stress svrs
  (2,42,82,45,92,95 only)
c = 10 - section data after FiberSIM conversion
  (shell181 only)
c = 11-17 - spares (note that the first three items in svindx are not available)
c nsvr (int,sc,inout) - number of dp words expected in this set

c output arguments:
c nsvr (int,sc,inout) - number of dp words in this set
c svr (dp,ar(nsvr),in) - data in this set

6.2.4. Subroutine svrput (Writing an Element’s Saved Variable Set)

*deck,svrput
subroutine svrput (svindx,nset,leng,svr)
c *** primary function: write out a svr data set for an element
c *** Notice - This file contains ANSYS Confidential information ***
c input arguments:
c svindx (int,ar(20),inout)- the index for svr for this element (see svgidx)
c nset (int,sc,in) - the set number in this index (same as svrget)
  - 1 - structural svrs
  - 2 - thermal/electric/fluid svrs
  - 3 - magnetic svrs
  - 4 - nonlinear svrs
  - 5 - plasticity svrs
  - 6 - creep svrs
  - 7 - coupled svrs
  - 8 - user svrs
  - 9 - initial stress svrs
  (2,42,82,45,92,95 only)
c = 10 - section data after FiberSIM conversion
  (shell181 only)
c = 11-17 - spares (note that the first three items in svindx are not available)
c leng (int,sc,in) - number of dp words in this set
6.2.5. Subroutine svpidx (Writing the Saved Variable Element Index to a File)

*deck,svpidx
subroutine svpidx (locsvr,svindx)
c *** primary function: write the svr element index onto file
c *** secondary functions: update the locsvr pointer to next element
c *** Notice - This file contains ANSYS Confidential information ***
c input arguments:
c locsvr (LONGINT,sc,inout) - pointer to start of svr for element
c svindx (int,ar(10,2),in) - index to svr for this element
   low and high parts of 64 bit address
c output arguments:
c locsvr (LONGINT,sc,inout) - pointer to start of svr for next element

6.2.6. Subroutine mreuse (Determining Which Element Matrices Can Be Reused)

*deck,mreuse
subroutine mreuse (kelrqk,kelfil,elem,ielc,kmasrt,knlmg,kconve,
x kpheno,kprop,nprop,prop,propo,krvro,rvr,rvro,amodo,asymo, kelin)
c *** primary function:
c determine which Matrices can be REUSEd and which must be recomputed
c from iteration to iteration.
c Note: a few special elements have some supplementary logic
c to adjust these results further. No attempt as been made to
c include all such logic in these routines.
c
Second note: this logic is essentially the same as the old
c sfrm logic. Hopefully, further simplifications and enhancements
c will be made in the future. (Especially in gap elements and in
c multilayer elements)
c the whole idea of kpheno, a holdover from the sfrm routines,
c needs to be looked at and possibly eliminated.
c
*** Notice - This file contains ANSYS Confidential information ***
c input arguments:
c kelrqq (int,ar(10),in) - request keys (needed for this analysis)
c kelfil (int,ar(10),in) - keys indicating matrices on the file
c elem (int,sc,in) - element number
c ielc (int,ar(IELCSZ),in) - array of element type characteristics
c kmasrt (int,sc,in) - does the mass matrix have rotational DOF?
   0 - no 1 - yes(with nlgeom, sfrm1n)
c knlmg (int,sc,in) - nonlinear magnetic curve exists in this
element
   0 - no 1 - yes
c kconve (int,sc,in) - key indicating existence of convections
   in this element
   0,1 - no 2 or more - yes
   must be input as 'i' if not used, as is
   changed in this routine(for analyzer).
c i - 0 must be used in calling routine
   if kpheno = 1.
c kpheno (int,sc,in) - key for type of phenomenon/level of check
   0 - structural like old sfrm1n,1s,3n,3s,f1
   1 - thermal like old sfrmic,1t,2t,3t
   2 - electrical/magnetic like some of old
   sfrmpo
   3 - general like old sfrmo
   c kprop (int,sc,in) - key indicating which material properties
   in the prop vector that need to be
   checked (see below)
6.2.7. Subroutine subrd (Reading Element Load Data for a Substructure Generation Run)

```c
*deck,subrd
  subroutine subrd (iel,key,nd,vect,ka)
  c *** primary function: read element load data from file for substructure
c generation run
  c *** secondary functions: none
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
    iel     (int,sc,in)     - element number
    key     (int,sc,in)     - type of load data
    = 1 temperature
    = 2 fluences
    = 3 heat generation rates
    = 4 current densities
    =10 pressures
    =11 film coefficients
    =12 bulk temperatures
    =13 extra displacement shapes
    =14 thermal strains(eptho in el42)
    =15 thermal flux (as in el55)
    =16 initial strains(epino in e101)
    =17 magnetic virtual displacements
    =18 calculated source field(hsn in e196)
    =20 element load vector
    =30 copy - do not scale(tempev in el42)
    =31 first load step only
    nd      (int,sc,in)     - number of data items
  c output arguments:
    propo   (dp,ar(nprop),inout)- current material properties
    rvro    (dp,ar(*),inout)   - current real constants
    amodo   (dp,sc,inout)     - current value of mode
    asymo   (dp,sc,inout)     - current value of isym
    kelin   (int,ar(10),out)  - keys indicating matrices to form
```
6.2.8. Subroutine subwrt (Writing an Element Load Vector to a File for a Substructure Generation Run)

*deck,subwrt
  subroutine subwrt (iel,nvect,key,nd,vect,ref)
  c *** primary function: write element load vect to file for substructure generation run
  c *** secondary functions: none
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c    iel      (int,sc,in)       - element number
  c    nvect    (int,sc,in)       - number of load vectors
  c              (current load step number)
  c    key      (int,sc,in)       - type of load vect
  c               = 1 temperature
  c               = 2 fluences
  c               = 3 heat generation rates
  c               = 4 current densities
  c               =10 pressures
  c               =11 film coefficients
  c               =12 bulk temperatures
  c               =13 extra displacement shapes
  c               =14 thermal strains(eptho in el42)
  c               =15 thermal flux (as in el55)
  c               =16 initial strains(epino in el01)
  c               =17 magnetic virtual displacements
  c               =18 calculated source field(hsn in el96)
  c               =20 element load vector
  c               =30 copy - do not scale(tempev in el42)
  c    nd       (int,sc,in)       - number of vect items
  c    vect     (dp,ar(nd),in)    - array of load data
  c    ref      (dp,sc,in)        - reference value for zero load
  c output arguments: none

6.2.9. Subroutine rvrget (Fetching Real Constants for an Element)

*deck,rvrget
  subroutine rvrget (iel,ireal,ielc,nrvr,rvr)
  c *** primary function: get the real constants for an element
  c     variable (typ,siz,intent)     description
  c        iel      (int,sc,in)       - element number
  c        ielc     (int,ar(*),in)    - real constant set number
  c output arguments:
  c    nrvr     (int,sc,out)      - number of real variables
  c    rvr      (dp,ar(*),out)     - element real constants

6.2.10. Subroutine propev (Evaluating a Group of Material Properties)

*deck,propev
  subroutine propev (iel,mtr,lp,tem,prop,n)
  c *** primary function: to evaluate a group of material properties
propev is used to pass two or more material property numbers thru the lp array to determine which temperature dependent material properties are to be evaluated. thus, the 3 prope1 calls:

call prope1 (elem,mat, 1,tem,e(1))
call prope1 (elem,mat,10,tem,alpha)
call prope1 (elem,mat,13,tem,dens)

should be combined as:

integer lp(3)
data lp /1,10,13/
call propev (elem,mat,lp(1),tem,prop(1),3)

*** Notice - This file contains ANSYS Confidential information ***

input arguments:
iel (int,sc,in) - element number
mtr (int,sc,in) - material number(input quantity mat, mat comma
lp (int,ar(n),in) - keys for which specific value is requested
each group must be in ascending order (ex,ey,ez, etc)
if negative, a required property
if zero, leave prop term unchanged

---- MP command labels --------
EX  = 1, EY  = 2, EZ  = 3, NUXY= 4, NUYZ= 5, NUXZ= 6, GXY = 7, GYZ = 8,
GXZ = 9, ALPX=10, ALPY=11, ALPZ=12, DENS=13, MU  =14, DAMP=15, KXX =16,
KYY =17, KZZ =18, RSVX=19, RSVY=20, C =22, HF =23, VIS=24,
EMIS=25, ENTH=26, LSST=27, PRXY=28, PRYZ=29, PRXZ=30, MURX=31, MURY=32,
MURZ=33, PERX=34, PERY=35, PERZ=36, MGXX=37, MGYY=38, MGZZ=39, EGXX=40,
EGGY=41, EGZZ=42, SBKX=43, SBKY=44, SBKZ=45, SONC=46, SLIM=47, ELIM=48,
USR1=49, USR2=50, USR3=51, USR4=51, FLUI=53, ORTH=54, CABL=55, RIGI=56,
HGLS=57, BM =58, QRAT=59, REFT=60, CTEX=61, CTEY=62, CTEZ=63, THSX=64,
THSY=65, THSZ=66, DMPR=67, LSSM=68, -69, -79, -71, -72,
-73, -74, -75, -76, -77, -78, -79, -80
(see mpinit for uncommented code)
(see chapter 2 of the elements volume of the user's manual
for a detailed description)

tem (dp,sc,in) - temperature at which to evaluate material
n (int,sc,in) - number of properties to be evaluated.
(20 maximum)
If n = 1, use propel instead.

output arguments:
prop (dp,ar(n),out) - values of material property

6.2.11. Subroutine prope1 (Evaluating One Material Property)
6.2.12. Subroutine pstev1 (Evaluating Material Properties for 1-D Elements)

```
*deck,pstev1
subroutine pstev1 (elem,matin,tem,prop)
  c *** primary function: to evaluate material properties for 1-d elements
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  c input arguments:
  c     elem     (int,sc,in)     - element number (for anserr)
  c     matin    (int,sc,in)     - material reference number
  c     tem      (dp,sc,in)      - temperature for evaluation
  c
  c output arguments:
  c     prop     (dp,ar(5),out)  - material properties: ex,nuxy,gxy,alpx,dens
```

6.2.13. Subroutine tbuser (Retrieving User Table Data)

```
*deck,tbuser
subroutine tbuser (mat,numitm,tbprop)
  c *** primary function: return the tb data for the user table
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  c input arguments:
  c     mat      (int,sc,in)     - material property number
  c     numitm   (int,sc,in)     - the number of data items requested
  c
  c output arguments:
  c     tbprop   (dp,ar(numitm),out) - array of tb data
```

6.2.14. Subroutine plast1 (Updating an Element’s Plastic History)

```
*deck,plast1
subroutine plast1 (option,elem,intpt,mat,kstartL,tem,dtem,e,
  x                  epeq,plwork,sigepl,sigrat,et)
  c *** primary function: to update the plastic history (for 1 component)
  c *** secondary functions: to compute the material tangent matrix if requested
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  c input arguments:
  c     option   (int,sc,in)     - plasticity option
  c     elem     (int,sc,in)     - element number (label)
  c     intpt    (int,sc,in)     - element integration point number
  c     mat      (int,sc,in)     - material reference number
  c     kstartL  (intL,sc,in)    - virtual starting address of the data table
  c     tem      (dp,sc,in)      - temperature at the end of this substep
  c     dtem     (dp,sc,in)      - temperature increment over this substep
  c     e        (dp,sc,in)      - elastic modulus
  c     epeq     (dp,sc,in)      - request key for tangent matrix formation
```
6.2.15. Subroutine plast3 (Updating an Element's Plastic History, 4 or 6 components)

Input arguments:
- option (int,sc,in) - plasticity option
- elem (int,sc,in) - element number (label)
- intpt (int,sc,in) - element integration point number
- mat (int,sc,in) - material reference number
- kstartL (intL,sc,in) - virtual starting address of the data table
- ncomp (int,sc,in) - number of stress/strain components (4 or 6)
- tem (dp,sc,in) - temperature at the end of this substep
- dtem (dp,sc,in) - temperature increment over this substep
- prop (dp,ar(9),in) - material property array (ex,ey,ez, gxy,gyz,gxz, uxy,uyz,uxz)
- d (dp,ar(ncomp,ncomp),in) - elastic stress-strain matrix
- ktform (int,sc,in) - request key for tangent matrix formation
- dens (dp,sc,in) - material density
- flu (dp,sc,in) - fluence at the end of this substep
- dflu (dp,sc,in) - fluence increment over this substep
- epel (dp,ar(ncomp),inout) - modified total strain (trial strain)
- eppl (dp,ar(ncomp),inout) - plastic strain at previous substep
- statev (dp,ar(ncomp,6),inout) - state variables at previous substep
- usvr (dp,ar(*),inout) - user-defined state variables (for userpl)
- epeq (dp,sc,inout) - effective plastic strain at prev substep
- plwork (dp,sc,inout) - accumulated plastic work at prev substep
- sigepl (dp,sc,out) - stress value on stress-strain curve
- sigrat (dp,sc,out) - ratio of trial stress to yield stress
- et (dp,sc,out) - tangent modulus
- cmel (int,sc,in) - elastoplastic key (form dtt if kplst=1)

Output arguments:
- epel (dp,ar(ncomp),inout) - elastic strain
- eppl (dp,ar(ncomp),inout) - updated plastic strain
- statev (dp,ar(ncomp,6),inout) - updated state variables
- usvr (dp,ar(*),inout) - updated user-defined state variables
- epeq (dp,sc,inout) - updated effective plastic strain
- plwork (dp,sc,inout) - updated accumulated plastic work

Internal variables:
- deppl (dp,sc) - equivalent plastic strain increment
6.2.16. Subroutine creep1 (Updating an Element's Creep History)

*deck, creep1
  subroutine creep1 (option, elem, intpt, mat, kstartL, epel, e, epcrp,
    x statev, usvr, tem, dtem, fluen, dflu, sig)
  c *** primary function: to update the creep history for 1-d elements
  c       used by: LINK1, LINK8, BEAM23, BEAM24, and
  c           SOLID65 (reinforcing)
  c *** Notice - This file contains ANSYS Confidential information ***

  c input arguments:
  c     option   (int, sc, in)         - creep option
  c     elem     (int, sc, in)         - element number (label)
  c     intpt    (int, sc, in)         - element integration point number
  c     mat      (int, sc, in)         - material reference number
  c     kstartL  (intL, sc, in)        - virtual starting address of the data table
  c     epel     (dp, sc, inout)       - elastic strain
  c     e        (dp, sc, in)          - elastic modulus
  c     epcrp    (dp, sc, inout)       - creep strain at previous substep
  c     statev   (dp, ar(7), inout)    - state variables at previous substep
  c     usvr     (dp, ar(*), inout)    - user-defined state variables (for usercr)
  c     tem      (dp, sc, in)          - temperature at the end of this substep
  c     dtem     (dp, sc, in)          - temperature increment over this substep
  c     fluen    (dp, sc, in)          - fluence at the end of this substep
  c     dflu     (dp, sc, in)          - fluence increment over this substep
  c     epel     (dp, sc, inout)       - elastic strain adjusted for creep increment
  c     sig      (dp, sc, inout)       - stress (not really used)

  c output arguments:
  c     epcrp    (dp, sc, inout)       - updated creep strain
  c     statev   (dp, ar(7), inout)    - updated state variables
  c     usvr     (dp, ar(*), inout)    - updated user-defined state variables
  c     sig      (dp, sc, inout)       - stress (recomputed if requested)

6.2.17. Subroutine creep3 (Updating an Element's Creep History, 3-D Elements)

*deck, creep3
  subroutine creep3 (option, elem, intpt, mat, kstartL, ncomp, epel, e,
    x posn, d, epcrp, statev, usvr, tem, dtem, fluen, dflu, kplst, sig, hsig)
  c *** primary function: to update the creep history for 3-d elements
  c       used by: PLANE02, PLANE13, PIPE20, PLANE42, SHELL43, SOLID45,
  c           SHELL51, PIPE60, SOLID62, SOLID65, PLANE62, SHELL91,
  c           SOLID92, SHELL93, SOLID95, SHELL143, SOLID191
  c *** Notice - This file contains ANSYS Confidential information ***

  c input arguments:
  c     option   (int, sc, in)         - creep option
  c     elem     (int, sc, in)         - element number (label)
  c     intpt    (int, sc, in)         - element integration point number
  c     mat      (int, sc, in)         - material reference number
  c     kstartL  (intL, sc, in)        - virtual starting address of the data table
  c     ncomp    (int, sc, in)         - number of stress/strain components (4 or 6)
  c     epel     (dp, ar(ncomp), inout) - elastic strain
  c     e        (dp, sc, in)          - elastic young'S MODULUS
  c     posn     (dp, sc, in)          - poisson'S RATIO
  c     d        (dp, ar(ncomp,ncomp), in) - elastic stress-strain matrix
6.2.18. Subroutine swell1 (Updating an Element's Swelling History)

*deck,swell1
subroutine swell1 (option,elem,intpt,mat,kstartL,epswel,epel,e,
x fluen,dfluen,tem,dtem,usvr)
c *** primary function: to update the swelling history for 1-d elements
     used by: LINK1, LINK8, BEAM23, and BEAM24

c *** Notice - This file contains ANSYS Confidential information ***

c input arguments:
c option (int,sc,in) - swelling option
c elem (int,sc,in) - element number (label)
c intpt (int,sc,in) - element integration point number
c mat (int,sc,in) - material reference number
c kstartL (intL,sc,in) - virtual starting address of the data table
c epswel (dp,sc,inout) - swell strain at previous substep
c epel (dp,sc,inout) - elastic strain
     e (dp,sc,in) - elastic young'S MODULUS
c fluen (dp,sc,in) - fluence at the end of this substep
c dfluen (dp,sc,in) - fluence increment over this substep
c tem (dp,sc,in) - temperature at the end of this substep
c dtem (dp,sc,in) - temperature increment over this substep
c usvr (dp,ar(*),inout) - user-defined state variables (for usersw)

c output arguments:
c epel (dp,sc,inout) - elastic strain adjusted for swelling inc
c epswel (dp,sc,inout) - updated swelling strain
c usvr (dp,ar(*),inout) - updated user-defined state variables

6.2.19. Subroutine swell3 (Updating an Element's Swelling History, 3-D Elements)

*deck,swell3
subroutine swell3 (option,elem,intpt,mat,kstartL,ncomp,epswel,
x epel,e,nuxy,fluen,dfluen,tem,dtem,usvr)
c *** primary function: to update the swelling history for 3-d elements
     used by: PLANE02, PLANE13, PIPE20, PLANE42, SHELL43, SOLID45,
            SHELL51, PIPE60, SOLID62, PLANE82, SHELL91, SOLID92,
            SHELL93, SOLID95, SHELL143, SOLID191

c *** Notice - This file contains ANSYS Confidential information ***

c input arguments:
c option (int,sc,in) - swelling option
c elem (int,sc,in) - element number (label)
c intpt (int,sc,in) - element integration point number
c mat (int,sc,in) - material reference number
c kstartL (intL,sc,in) - virtual starting address of the data table
c ncomp (int,sc,in) - number of stress/strain components (4 or 6)

output arguments:
c epel (dp,sc,inout) - elastic strain adjusted for swelling inc
c epswel (dp,sc,inout) - updated swelling strain
c usvr (dp,ar(*),inout) - updated user-defined state variables
6.2.20. Function eLenPsvrBuf (Determining additional ESAV Record for Plasticity)

*deck,eLenPsvrBuf
function eLenPsvrBuf (mat, plOpt, ncomp)

*** primary function: determine additional esave record for plasticity

*** input arguments
mat (int,sc,in) - material ID
plOpt (int,sc,in) - plasticity option
ncomp (int,sc,in) - number of strain components (1, 4, or 6)

*** output arguments
elLenPsvrBuf (int,sc,out) - number of extra data items saved

*** local variables

6.2.21. Function nlget (Retrieving Material Nonlinear Property Information)

*deck, nlget
function nlget (mat,iprop,prop)

*** primary function: get a material non-linear property (TB) table.

*** Notice - This file contains ANSYS Confidential information ***

*** input arguments:
mat (int,sc,in) - material number
iprop (int,sc,in) - property number (tbpnum in tblecm)

*** output arguments:
nlget (int,sc,out) - number of property values
prop (dp,ar(nlget),out) - vector of the property values

--- terms of the descriptor record:
header(1) = tbtyp
header(2) = tbtems
header(3) = temloc
header(4) = dprtem
6.2.22. Subroutine usereo (Storing Data in the nmisc Record)

*deck,usereo                                                                 user
  subroutine usereo (elem,iout,nbsvr,bsvr,nnrsvr,nrsvr,npsvr,psvr,
      x ncsvr,csvr,nusvr,usvr,nnode,nodes,xyz,vol,leng,time,
      x timinc,nutot,utot,maxdat,numdat,udbdat)

** primary function: to call userou, which allows user to store
data in nmisc record

** Notice - This file contains ANSYS Confidential information ***

input arguments:
- variable (typ,siz,intent) description
- elem (int,sc,in) - element number
- iout (int,sc,in) - output unit number
- nbsvr (int,sc,in) - number of basic element variables
- bsvr (dp,ar(nbsvr),in) - basic element variables
- nnrsvr (int,sc,in) - number of nonlinear element variables
- nrsvr (dp,ar(nnrsvr),in) - nonlinear element variables
- npsvr (int,sc,in) - number of plasticity element variables
- psvr (dp,ar(npsvr),in) - plasticity element variables
- ncsvr (int,sc,in) - number of creep element variables
- csvr (dp,ar(ncsvr),in) - creep element variables
- nusvr (int,sc,in) - number of user-supplied element variables
- usvr (dp,ar(nusvr),in) - user-supplied element variables
- nnode (int,sc,in) - number of nodes
- nodes (int,ar(nnode),in) - node numbers
- xyz (dp,ar(6,nnode),in) - nodal coordinates and rotations (virgin)
- vol (dp,sc,in) - element volume (or area if 2-d)
- leng (dp,sc,in) - element length (beams,spars,etc)
- time (dp,sc,in) - current time
- timinc (dp,sc,in) - current sub step time increment
- nutot (int,sc,in) - length of dof solution vector utot
- utot (dp,ar(nutot),in) - solution vector
- maxdat (int,sc,in) - size of user output array (3 x nnode)
  actually, = ielc(nmmup)

output arguments:
- variable (typ,siz,intent) description
- numdat (int,sc,out) - number of user output items in array udbdat
- udbdat (dp,ar(maxdat),out) - user output items to be placed at the end
of the nmisc record

6.2.23. Subroutine eldwrtL (Writing Element Data to a File)

*deck,eldwrtL     subroutine eldwrtL (ielem,edtype,lcerstL,edindxL,nval,value)
  *** primary function: output element data to result file.

** Notice - This file contains ANSYS Confidential information ***

input arguments:
- ielem (int,sc,in) - element number
- edtype (int,sc,in) - element data type (see elparm)
6.2.24. Subroutine eldwrnL (Writing Element Nonsummable Miscellaneous Data to the Results File)

*deck, eldwrnL
  subroutine eldwrnL (elem, ielc, lcerstL, edindxL, nudb, udbdat, 
                      nval, value, ndval)
  c *** primary function: output element nonsummable miscellaneous data
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c elem (int, sc, in) - element number
  c ielc (int, ar(IELCSZ), in) - element characteristic vector
  c defined in elccmt
  c lcerstL (LONG, sc, inout) - pointer to results file position
  c edindxL (LONG, ar(25), inout) - index to results file data
  c nudb (in, sc, inout) - size of what the user wants to add
  c udbdat (dp, ar(*), in) - what the user wants to add
  c nval (int, sc, in) - the total number of values to
                      be output (does not include nudb)
  c value (dp, ar(nval), in) - output values
  c ndval (int, sc, in) - dimension of value - must be no less than
                         ielc(NMNMIS) + ielc(NMNMUP)

6.2.25. Subroutine trrot (Computing the Rotation Vector)

*deck, trrot
  subroutine trrot (tr, rot)
  c *** primary function: get the rotation vector from a transformation matrix
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c tr (dp, ar(3,3), in) - transformation matrix
  c output arguments:
  c rot (dp, ar(3), out) - rotation vector

6.2.26. Subroutine rottr (Computing the Transformation Matrix)

*deck, rottr
  subroutine rottr (rot, tr)
  c primary function: compute transformation matrix from rotation vector *****
  c *** Notice - This file contains ANSYS Confidential information ***
  c ref(new): eqn. (b.4), simo and vu-quoc, camme, 58 (1986), 79-116
  c (removes singularities at pi and 2*pi)
  c input arguments:
  c variable (typ, siz, intent) description
  c rot (dp, ar(4), in) - rotation parameter in radians
  c output arguments:
6.2.27. Subroutine xyzup3 (Updating an Element's 3-D Nodal Coordinates)

*deck,xyzup3
  subroutine xyzup3 (nnod,u,xyz,nx,xyzup)
  c *** primary function: update a 3-d ele nodal coords for large deformation
  c
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  c input arguments:
  c    nnod    (int,sc,in)          - number of nodes
  c    u        (dp,ar(nr),in)       - displacement vector
  c    nr       (int,sc,in)          - size of the u vector
  c    xyz      (dp,ar(nx,nnod),in)  - coordinates to be updated
  c    nx       (int,sc,in)          - row size of xy
  c
  c output arguments:
  c    xyzup     (dp,ar(3,nnod),out) - updated coordinates
  c

6.2.28. Subroutine updrot (Updating the Rotation Pseudovector)

*deck,updrot
  subroutine updrot (v2,w1)
  c primary function: update the rotation pseudovector for 3-d large rotations *****
  c
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  c The updating of the pseudovector uses the mathematics of quaternions
  c (ref: eqn. a5 of J. H. Argyris, CMAME, 32(1982)85-155). The pseudovector uses the normalization proposed by Rankin and Brogan (ref:
  c eqn. 15, JPVT, 108(1986)165-174).
  c CMAME = Computer Methods in Applied Mechanics and Engineering
  c JPVT = Journal of Pressure Vessel Technology (ASME)
  c
  c variable descriptions:
  c    input:
  c v2     - rotation increment
  c w1     - previous rotation pseudovector
  c    output:
  c w1     - updated pseudovector
  c
  c v1 = cos(v1/2) + 1/2*w1,   w1 = 2*sin(v1/2)*e1
  c v2 = cos(v2/2) + 1/2*w2,   w2 = 2*sin(v2/2)*e2
  c v21 = v2*v1 = cos(v21/2) + 1/2*w21 (quarternion multiplication)
  c w1 =: v21  (w1 is updated)
  c

6.2.29. Subroutine tmptget (Defining Current Temperature Loads)

*deck,tmptget
  subroutine tmptget (iel,ielc,nnod,nodes,ref,ndat0,begdat,dat, x   enddat,tlvf)
  c primary function: define the current temperature loads
  c
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  c typ=int,dp,log,chr,dcp   siz=ar(n),func    intent=in,out,inout
  c
  c input arguments:
  c variable (typ,siz,intent)     description
  c iel    (int,sc,in)         - element number
  c ielc   (int,ar(IELCSZ),in) - array of element type characteristics
  c nnod   (int,sc,in)         - number of nodes in the nodes array
  c nodes  (int,ar(nnod),in)   - list of nodes
  c ref    (dp,sc,in)          - reference temperature
c ndat (int, sc, in) - number of data items to get
b c begdat (dp, ar(ndat), in) - data at the beginning of this load step
c
output arguments:
c dat (dp, ar(ndat), out) - data at this time point
c enddat (dp, ar(ndat), out) - data at end of this load step
c tlvf (int, sc, out) - thermal load vector flag

c Should the thermal load vector be computed

c = 0 - no, temperatures match tref

c = 1 - yes, temperatures do not match tref

Note, that even if tlvf = 0, temperatures may be used to
compute temperature-dependent material properties.

6.2.30. Subroutine prsget (Defining Current Pressure Loads)

*deck, prsget
subroutine prsget (iel, ielc, nfac, ndat, begdat, dat, enddat, iexist)

primary function: define the current pressure loads

See also: PrsRIGet

input arguments:

c iel (int, sc, in) - element number

c ielc (int, ar(IELCSZ), in) - array of element type characteristics

c nfac (int, sc, in) - number of pressure faces

c ndat (int, sc, in) - number of pressure values

c begdat (dp, ar(ndat), in) - pressure at the beginning of load step

output arguments:

c dat (dp, ar(ndat), out) - pressures at this iteration

c enddat (dp, ar(ndat), out) - pressure at end of this load step

c iexist (int, sc, out) - flag if pressure exist

= 0 - no pressure

= 1 - yes pressure

6.2.31. Subroutine cnvget (Defining Current Convection Loads)

*deck, cnvget
subroutine cnvget (iel, ielc, nr, u, nfac, ndat, beghc, begtb, 
x hc, tb, endhc, endtb, iexist)

primary function: define the current convection loads

input arguments:

c iel (int, sc, in) - element number

c ielc (int, ar(IELCSZ), in) - array of element type characteristics

c nr (int, sc, in) - dimension of u (temperature) vector

c u (dp, ar(nr), in) - most current temperatures

c nfac (int, sc, in) - number of convection faces

c ndat (int, sc, in) - number of convection values

c beghc (dp, ar(ndat), in) - hcoef at the beginning of load step

c begtb (dp, ar(ndat), in) - tbulk at the beginning of load step

output arguments:

c hc (dp, ar(ndat), out) - hcoef at this substep

c tb (dp, ar(ndat), out) - tbulk at this substep

c endhc (dp, ar(ndat), in) - hcoef at the end of this load step

c endtb (dp, ar(ndat), in) - tbulk at the end of this load step

c iexist (int, sc, out) - flag if convection exist

= 0 - no convection

= 1 - constant convection (with time)

= 2 - changing convection (with time)
6.2.32. Subroutine hgnget (Defining Current Heat Generation Loads)

*deck, hgnget

  subroutine hgnget (iel, ielc, nnod, nodes, ndat, begdat, dat, enddat, 
    x      iexist)
  c     primary function: define the current heat generation loads
  c
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  typ=int, dp, log, chr, dcp    siz=sc, ar(n), func    intent=in, out, inout
  c
  input arguments:
  c variable (typ, siz, intent)     description
  c iel   (int, sc, in)        - element number
  c ielc  (int, ar(IELCSZ), in) - array of element type characteristics
  c nnod  (int, sc, in)        - number of nodes in the nodes array
  c nodes (int, ar(nnod), in)   - list of nodes
  c ndat  (int, sc, in)        - number of data items to get
  c begdat (dp, ar(nadat), in) - data at the beginning of this load step
  c
  output arguments:
  c dat   (dp, ar(ndat), out)  - data at this time point
  c enddat (dp, ar(ndat), out) - data at end of this load step
  c iexist (int, sc, out)     - flag if heat generation exist
  c = 0 - no heat generation
  c = 1 - yes heat generation
  c

6.2.33. Subroutine prinst (Computing principal stress and stress intensity)

*deck, prinst

  subroutine prinst (s)
  c    primary function: computes principal stresses and stress intensity
  c    secondary functions: none
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  input arguments:
  c variable (typ, siz, intent)     description
  c s     (dp, ar(11), inout)   - stress vector
  c s(1)  = sx
  c s(2)  = sy
  c s(3)  = sz
  c s(4)  = sigxy
  c s(5)  = sigyz
  c s(6)  = sigzx
  c
  output arguments:
  c variable (typ, siz, intent)     description
  c s     (dp, ar(11), inout)   - stress vector
  c s(7)  = sig1
  c s(8)  = sig2
  c s(9)  = sig3
  c s(10) = s.i.
  c s(11) = sige
  c

6.3. Routines for Modifying and Monitoring Existing Elements

The next few pages describe the user routines you use to modify or monitor existing ANSYS elements. These
routines enable you to perform tasks including:

• Computing load vectors for frequency domain logic
6.3.1. Subroutine userfd (Computing the Complex Load Vector for Frequency Domain Logic)

*deck, userfd
subroutine userfd (nr, kcbbrm, kpfor, ktrsur, isur,
    x cb, do, doext, area, alenv, denswat, faclen, conac, fluidt, visc,
    x watbas, watcur, watwav, xyzup, tr, accel, puvel, u, zass,
    x forl, zsc, zsc2, pdyn, holdwv)

C *** primary function: compute complex load vector for frequency domain logic
C for pipe59
C *** secondary functions: none
C -- accessed with keyopt(12) = 2
C
C *** copyright(c) 2006 SAS IP, Inc. All rights reserved.
C *** ansys, inc.
C *** Notice - This file contains ANSYS Confidential information ***
C
C input arguments:
C nr (int, sc, in) - matrix size
C kcbbrm (int, sc, in) - key for reduced matrices/cable option
C kpfor (int, sc, in) - keyopt for hydrodynamic printout
C ktrsur (int, sc, in) - keyopt for surface treatment (unfinished)
C isur (int, sc, in) - surface flag
C cb (dp, sc, in) - buoyancy coefficient (real constant)
C do (dp, sc, in) - outside diameter of pipe
C doext (dp, sc, in) - outside diameter of insulation
C area (dp, sc, in) - area of displaced water
C alenv (dp, sc, in) - length of element
C denswat (dp, sc, in) - water density
C faclen (dp, sc, in) - wetted fraction of pipe
C conac (dp, sc, in) - added mass per unit length
C fluidt (dp, sc, in) - fluid temperature
C visc (dp, sc, in) - viscosity
C watbas (dp, ar(*), in) - water basic table
C watcur (dp, ar(*), in) - water current table
C watwav (dp, ar(*), in) - water wave table
C xyzup (dp, ar(3, 2), in) - updated coordinates
C tr (dp, ar(3, 3), in) - local to global transformation matrix
C accel (dp, ar(3), in) - acceleration vector
C puvel (int, sc, in) - index for velocities in u matrix
C u (dp, ar(nr, 5), in) - displacements and velocities
C zass (dp, ar(nr, nr), in) - mass matrix
C forl (dp, ar(12), inout) - force vector in element coordinates
C zsc (dp, ar(nr), inout) - real load vector for frequency domain
C zsc2 (dp, ar(nr), inout) - complex load vector for frequency domain
C
C output arguments:
C forl (dp, ar(12), inout) - force vector in element coordinates
C zsc (dp, ar(nr), inout) - real load vector for frequency domain
C zsc2 (dp, ar(nr), inout) - complex load vector for frequency domain

- Storing element output that users supply
- Modifying the orientation of material properties and stresses
- Modifying the orientation of material properties and stresses of layers within an element
- Performing a user-defined operation on a parameter for the COMBIN7 and COMBIN37 elements
- Providing a user-defined initial thickness for SHELL181
- Providing a user-defined initial stress for PLANE2, PLANE42, SOLID45, PLANE82, SOLID92, SOLID95, LINK180, SHELL181, PLANE182, PLANE183, SOLID185, SOLID186, SOLID187, SOLSH190, BEAM188, BEAM189, SHELL208, and SHELL209
- Providing a user-defined fictive temperature relationship for VISCO88 and VISCO89
- Providing viscoelastic computation in the stiffness pass for VISCO88 and VISCO89
- Modifying SURF151 and SURF152 film coefficients and bulk temperatures based on information from FLUID116
6.3.2. Subroutine userou (Storing User-Supplied Element Output)

*deck, userou
subroutine userou (elem, iout, nbsvr, bsvr, nnrsvr, nrsvr, npsvr, psvr,
x ncsvr, csvr, nusvr, usvr, nnode, nodes, xyz, vol, leng, time,
x timinc, nutot, utot, maxdat, numdat, udbdat)
c
*** primary function: store user supplied element output
in nmisc record
in order to activate this user programmable feature,
the user must enter the usrcal command.
c
*** copyright(c) 2006 SAS IP, Inc. All rights reserved.
*** ansys, inc.
c *** Notice - This file contains ANSYS Confidential information ***
c
c  input arguments:
c variable (typ, siz, intent)       description
  elem    (int, sc, in)     - element number
  iout    (int, sc, in)     - output unit number
  nbsvr   (int, sc, in)     - number of basic element variables
  bsvr    (dp, ar(nbsvr), in) - basic element variables
  nnrsvr  (int, sc, in)     - number of nonlinear element variables
  nrsvr   (dp, ar(nnrsvr), in) - nonlinear element variables
  npsvr   (int, sc, in)     - number of plasticity element variables
  psvr    (dp, ar(npsvr), in) - plasticity element variables
  ncsvr   (int, sc, in)     - number of creep element variables
  csvr    (dp, ar(ncsvr), in) - creep element variables
  nusvr   (int, sc, in)     - number of user-supplied element variables
  usvr    (dp, ar(nusvr), in) - user-supplied element variables
  nnode   (int, sc, in)     - number of nodes
  nodes   (int, ar(nnode), in) - node numbers
  xyz     (dp, ar(6,nnode), in) - nodal coordinates and rotations (virgin)
  vol     (dp, sc, in)      - element volume (or area if 2-d)
  leng    (dp, sc, in)      - element length (beams, spars, etc)
  time    (dp, sc, in)      - current time
  timinc  (dp, sc, in)      - current sub step time increment
  nutot   (int, sc, in)     - length of dof solution vector utot
  utot    (dp, ar(nutot), in) - solution vector
  maxdat  (int, sc, in)     - size of user output array (3 x nnode)

c  output arguments:
c variable (typ, siz, intent)       description
  numdat  (int, sc, out)          - number of user output items in array udbdat
  (maximum size of numdat is ielc(NMNMUP))
  which is usually three times the number
  of nodes.
  udbdat  (dp, ar(maxdat), out)   - user output items to be placed at the end
  of the nmisc record
6.3.3. Subroutine useran (Modifying Orientation of Material Properties)

```
*deck,useran
subroutine useran (vn,vref,elem,thick,xyzctr,bsangl)
c  user written routine to modify orientation of material properties
c and stresses ****************************
c  applicable to: shell43,63,91,93,99, solid46,64,191
c  accessed by keyopt

c *** copyright(c) 2006 SAS IP, Inc. All rights reserved.
c *** ansys, inc.
c **** Notice - This file contains ANSYS Confidential information ***
c
  if you do, your results are probably wrong.

input(do not change)---
c  vn   = vector normal to element
c  vref = unit vector orienting element, essentially edge i-j
c  elem = element number
c  thick = total thickness of element at this point (see note below)
c  xyzctr = location of element centroid or integration point

c output---
c  bsangl = output from this subroutine. it represents the angle(s)
c          between vref and the desired orientation. it may have
c          the default orientation coming in to useran.
c          This will be combined with the angles derived from
c          the ESYS command.
c          use 1 angle for 2-d elements and shells
          use 3 angles for 3-d solids
```

6.3.4. Subroutine usanly (Modifying Orientation of Material Properties and Stresses of Layers)

```
*deck,usanly
subroutine usanly (vn,vref,elem,thick,xyzctr,ln,bsangl)
c  user written routine to modify orientation of material properties
c and stresses of layers within an element ****************************
c  applicable to shell91,99, and solid46,191
c  accessed with keyopt(4)

c *** copyright(c) 2006 SAS IP, Inc. All rights reserved.
c *** ansys, inc.
c **** Notice - This file contains ANSYS Confidential information ***
c
  if you do, your results are probably wrong.

input(do not change)---
c  vn   = vector normal to element
c  vref = unit vector orienting element, essentially edge i-j
c  elem = element number
c  thick = thickness of layer ln at this point (see note below)
c  xyzctr = location of element centroid or integration point
c  ln   = layer number

c output---
c  bsangl = output from this subroutine. it represents the angle(s)
c          between vref and the desired orientation of the layer.
c          it may have the default orientation coming in to useran.
c          this angle does not affect the angle given in the printout.
c          users may want to add their own angle printout in usanly.
c          also, this angle does not affect the angle shown using the
c          layplot or laylist commands
          use 1 angle for 2-d elements and shells
```
6.3.5. Subroutine userrc (Performing User Operations on COMBIN7 and COMBIN37 Parameters)

```fortran
*deck, userrc
subroutine userrc (elem, ireal, type, nusvr, usvr, parm, parmld, x c1, c2, c3, c4, fcon)
c primary function: user operation on parameter for combin7 and combin37
accessed with keyopt(9) = 1

*** copyright(c) 2006 SAS IP, Inc. All rights reserved.
*** ansys, inc.
c *** Notice - This file contains ANSYS Confidential information ***
c
input arguments:
c variable (typ, siz, intent)       description
elem (int, sc, in)                - element number
ireal (int, sc, in)               - element real constant number
type (int, sc, in)                - element type number
nusvr (int, sc, in)               - number of user-supplied element variables
usvr (dp, ar(nusvr), inout)       - user-supplied element variables
parm (dp, sc, in)                 - current value of the parameter
c1 (dp, sc, in)                   - real constant c1
c2 (dp, sc, in)                   - real constant c2
c3 (dp, sc, in)                   - real constant c3
c4 (dp, sc, in)                   - real constant c4

output arguments:
c variable (typ, siz, intent)       description
usvr (dp, ar(nusvr), inout)       - user-supplied element variables
may be sent .rst file with usereo
fcon (dp, sc, out)                - result of calculation
c either c1 or c3 must be nonzero for this logic to be accessed,
c
```

6.3.6. Function userpe (Calculating Rotation Caused by Internal Pressure)

```fortran
*deck, userpe
function userpe (prs, rvrp, angle, ex, nuxy)
c primary function: calculate the rotation caused by internal pressure
on an elbow element
This function is only called by el18(pipe18)
if keyopt(5) = 1

*** copyright(c) 2006 SAS IP, Inc. All rights reserved.
*** ansys, inc.
c
input arguments:
c variable (typ, siz, intent)       description
prs (dp, ar(5), in)                - pressure vector
rvrp (dp, ar(11), in)              - real constants (see elements manual)
angle (dp, sc, in)                 - subtended angle
ex (dp, sc, in)                    - Young's modulus
nuxy (dp, sc, in)                  - Poisson's ratio

c output arguments:
c variable (typ, siz, intent)       description
userpe (dp, sc, out)               - rotation caused by internal pressure on the
```
6.3.7. Subroutine UEIMatx (Accessing Element Matrices and Load Vectors)

*deck,UEIMatx
subroutine UEIMatx (elem,nr,ls,zs,zsc,uelm,ielc,nodes,
 x                   ElDofEachNode,elmdat,xyzang,lenu)

   c primary function: User routine to access element matrices and load vectors.
   c Needs to have USRCAL,UELMATX to be accessed.
   c Called after the call to the element routine and
   c before the solver.
   c May be used to monitor and/or modify the element matrices
   c and load vectors.

   *** copyright(c) 2006 SAS IP, Inc. All rights reserved.
   *** ansys, inc.

   typ=int,dp,log,chr,dcp siz=sc,ar(n) intent=in,out,inout

   c input arguments:
   c variable (typ,siz,intent)       description
   c elem   (int,sc,in)          - User element number
   c nr     (int,sc,in)          - number of rows in element matrix
   c ls     (int,ar(nr),in)      - DoF Index vector for this element matrix
   c zs     (dp,ar(nr,nr,4),inout) - K,M,C,SS matrices for this element
   c zsc    (dp,ar(nr,2),inout)  - Element load vector and N-R correction vec
   c uelm   (dp,ar(nr,5),in)     - Nodal displacements for this element
   c ielc   (int,ar(4),in)       - Element type characteristics
   c nodes  (int,ar(*),in)       - Nodes for this element
   c ElDofEachNode (int,ar(nr),in) - list of dofs for each node in Global
   c elmdat (int,ar(10),in)      - Element data for this element
   c xyzang (dp,ar(6,*),in)      - X,Y,Z,THXY,THYZ,THZX for each element node
   c lenu   (int,sc,in)          - Length of global displacement vector

   c output arguments:
   c zs     (dp,ar(nr,nr,4),inout) - K,M,C,SS matrices for this element
   c zsc    (dp,ar(nr,2),inout)  - Element load vector and N-R correction vec
   c WARNING: any CHANGES to these (or any other) arguments will have a direc
   c impact on the solution, possibly giving meaningless results. The normal
   c usage of this routine is simply monitor what is happening.

6.3.8. Subroutine UTHICK (Getting User-defined Initial Thickness)

*deck,uthick
SUBROUTINE uthick (elemId, elemType, matId, realId,
 $                   numDomIntPts, curCoords, thickness)

   c *** primary function: get the user defined thickness
   c *** copyright(c) 2006 SAS IP, Inc. All rights reserved.
   c *** ansys, inc.

   c input arguments
   c Variable (type,sz,i/o)  description
   c elemId  (int,sc,i)     element number
   c elemType (int,sc,i)    element TYPE (181 etc.)
   c matId   (int,sc,i)     material number
   c realId  (int,sc,i)     real constant set number
   c numDomIntPts (int,sc,i) - number of integration points
   c curCoords (dp,ar(3,numDomIntPts),i) - current coordinates

   c output arguments
   c
6.3.9. Subroutine USTRESS (Getting User-defined Initial Stress)

```fortran
*deck,ustress                             user
  SUBROUTINE ustress (elemId, elemType, matId, basis,
                         curIntPt, curLayer, curSecPt,
                         numDirect, numShear, curCoords,
                         tLocation, iniData)

  *** primary function: get the user supplied stress state for
  initial stress analysis

  *** copyright(c) 2006 SAS IP, Inc. All rights reserved.
  *** ansys, inc.

  input arguments
  ===============
  Variable        (type,sz,i/o)  description
  elemId         (int,sc,i)     element number
  elemType       (int,sc,i)     element TYPE (42, 181 etc.)
  matId          (int,sc,i)     material number
  basis          (dp,ar(3,3),i) basis directions (for the element)
  curIntPt       (int,sc,i)     current Integration point
  curLayer       (int,sc,i)     current Layer number (for shell
                               and layered solids)
  curSecPt       (int,sc,i)     current point through the thickness in the
                               current layer
  numDirect      (int,sc,i)     number of direct stress comp
                              (s_xx,s_yy,s_zz)
  numShear       (int,sc,i)     number of shear stress comp
                              (s_xy,s_yz,s_zx)
  curCoords      (dp,ar(3),i)   current coordinates
  tLocation      (dp,sc,i)      normalized thickness location
                              (shell elements only)
                              -1 = bottom or first sec pt
                              +1 = top or last sec pt
                              0 = mid surface

  output arguments
  ================
  iniData        (dp,ar(numDirect+numShear),o)  user input stress values

  --- parameters

6.3.10. Subroutine UsrFictive (Providing User-defined Fictive Temperature Relationship)

```
**6.3.11. Subroutine UsrViscEl (Performs Viscoelastic Computation)**

```plaintext
*deck,UsrViscEl  user
  subroutine UsrViscEl (veinpt,ncomp,tem,dtem,ex,gxy,exx,exx,phil,
    x  zil,gil,g2l,hsm,hbm,smcm,bmcm,eps1,eps2,f1l,dfl,tref,rv,rc,rvd,rcd,
    x  dsg,rsig,cm,kerr,cml)

  ***** notice- this routine contains ansys, inc. confidential information *****

  *** copyright(c) 2006 SAS IP, Inc. All rights reserved.
  *** ansys, inc.

  input arguments:
    variable (type,sze,intent)    description
    veinpt (dp,ar(95),in)          - viscoelastic input data
    ncomp (int,sc,in)              - number of components (4 or 6)
    tem (dp,sc,in)                 - temperature
    dtem (dp,sc,in)                - incremental temperature
    ex (dp,sc,in)                  - bulk modulus at infinite time
    gxy (dp,sc,in)                 - shear modulus at infinite time
    smcm (dp,ar(ncomp,ncomp),in)   - material matrix for shear modulus
    bmcm (dp,ar(ncomp,ncomp),in)   - material matrix for bulk modulus
    eps1 (dp,ar(ncomp),in)         - strain for previous iteration
    eps2 (dp,ar(ncomp),in)         - total strain for current iteration
    tref (dp,sc,in)                - reference temperature

  output arguments:
    variable (type,sze,intent)    description
    eex (dp,sc,out)               - effective bulk modulus
    egxy (dp,sc,out)              - effective shear modulus
    phil (dp,sc,inout)            - previous shift factor
    zil (dp,sc,inout)             - previous pseudo time
    g1l (dp,ar(ncomp,10),out)     - recursive shear relaxation
    g2l (dp,ar(ncomp,10),out)     - recursive bulk relaxation
    hsm (dp,ar(10),out)           - recursive shear relaxation
    hbm (dp,ar(10),out)           - recursive bulk relaxation
    f1l (dp,sc,inout)             - previous fictive temperature
    dfl (dp,ar(10),out)           - incremental fictive temperature
    rv (dp,sc,inout)              - total volume change
    rvc (dp,sc,inout)             - incremental volume change
    dsg (dp,ar(ncomp),out)        - stress change
    rsig (dp,ar(ncomp),out)       - stress relaxation
    cm (dp,ar(ncomp,ncomp),out)   - total material matrix
    kerr (int,sc,sc)              - error key

  argument of convenience:
    cml (dp,ar(ncomp,ncomp),none)- no value (used only to avoid simplify
                             logic due to variable array sizes)
```
6.3.12. Subroutine usrsurf116 (Modifying SURF151 and SURF152 Film Coefficients and Bulk Temperatures)

*deck, usrsurf116
subroutine usrsurf116 (elem, ielc, center, jdim, kaxis, time, nr, u, temvel, hc, tb, key)
  c *** primary function: change element convection surface info
  c for surf151 and/or surf152 based on information from fluid116.
  c It is called by el151 and el152.
  c
  in order to activate this user programmable feature,
  the user must have used fluid116 with keyopt(2) = 1.
  Further, surf151 and/or surf152 must have keyopt(5) = 1
  (include extra node). Finally, for this routine to do anything,
  key(1) and/or key(2) must be reset in this routine to a
  nonzero number. There is no usrcal control over this routine.
  c
  *** copyright(c) 2006 SAS IP, Inc. All rights reserved.
  *** ansys, inc.
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  input arguments:
  c variable (typ, siz, intent) description
  c elem (int, sc, in) - element number for operation.
  c ielc (int, ar(IELCSZ), in) - array of element type characteristics
  c center (dp, ar(3), in) - coordinates of center of surface element
  c jdim (int, sc, in) - dimensionality key
  c kaxis (int, sc, in) - axis of rotation (keyopt(3) for el152)
  c time (dp, sc, in) - time of current substep
  c nr (int, sc, in) - number of nodal temperatures
  c u (dp, ar(nr), in) - vector of most recent values of the
  c temperatures
  c omeg (dp, sc, in) - spin real constant (may be from table)
  c ndat (int, sc, in) - number of data points per element
  c hc (dp, ar(ndat), inout) - film coefficients
  c (has input values for each corner
  c of element)
  c tb (dp, ar(ndat), inout) - bulk temperature
  c (has input values for each corner
  c of element)
  c
  output arguments:
  c variable (typ, siz, intent) description
  c temvel (dp, sc, out) - user defined bulk temperature in excess of
  c fluid node temperature
  c hc (dp, ar(ndat), inout) - film coefficients
  c (defines input values for each corner
  c of element)
  c tb (dp, ar(ndat), inout) - bulk temperature (includes any modification)
  c (defines input values for each corner
  c of element)
  c key (int, ar(2), out) - key if to use this logic
  c key(1) = 0 = no new film coefficient
  c key(1) = 1 = define new film coefficient
  c key(2) = 0 = no new bulk temperature
  c key(2) = 1 = define new bulk temperature
  c (if key(2) = 1, the adiabatic wall
  c temperature logic is not used).
  c
  this routine is called during each substep of each load step.
  it is called for each equilibrium iteration.
  it is called once per element. it is called only during the heat
  flow load vector formulation stage, and not during the heat flow
  evaluation stage.
  the call to get the standard ansys input convection surfaces
This routine may be thought of as a specialized version of usercv. Indeed, el151 and el152 also call usercv. Either (or both, rarely) could be used.

velocity-dependent film coefficients and bulk temperatures can be computed by using the velocities and other information from fluid116.

Details of this procedure are:
-- SURF151 or SURF152 are 'pasted' onto the actual solid model.
-- flow rate is input to or is computed by FLUID116, with KEYOPT(2) = 1
-- flow rate may be a function of time
-- the user defines nodes on the FLUID116 network to be the same nodes as the 'extra' nodes of SURF151 or SURF152. If more than one FLUID116 element is attached to one of these nodes, the velocities are averaged.
-- SURF151 or SURF152 calls this routine, indirectly, to compute the film coefficient and bulk temperature. This routine, in turn, gets the average velocity at the 'extra' node using 'getv116', as shown below. Other quantities brought in by getv116 are also averaged.

6.3.13. Subroutine User116Cond (Computes the conductance coefficient for FLUID116)

6.3.14. Subroutine User116Hf (Computes the film coefficient for FLUID116)
6.3.15. Subroutine Us_Surf_Str (Captures surface stresses)

```fortran
*deck,Us_Surf_Str                              user
  subroutine Us_Surf_Str (elem,face,area,temp,pressure,ep,stress)
  c *** primary function: User routine to capture surface stresses
  c *** called by PLANE2, 42, 82 and SOLID45, 92, 95
  c *** Notice - This file contains ANSYS Confidential information ***
  c *** copyright(c) 2006 SAS IP, Inc. All rights reserved.
  c *** ansys, inc.
  c input arguments:
  c variable (typ,siz,intent)    description
  c elem    (int,sc,in)       element number
  c face    (int,sc,in)       face number
  c area    (dp, sc,in)       face area (or length)
  c temp    (dp, sc,in)       face temperature
  c pressure(dp, sc,in)       face pressure
  c ep      (dp,ar(4),in)     face strains
  c stress  (dp,ar(11),in)    face stresses
  c output arguments: none
```

6.3.16. Subroutine usflex (Computes the flexibility factor for PIPE16, PIPE17, PIPE18, and PIPE60)

```fortran
*deck,usflex                              user
  subroutine usflex (etype,elem,rvrm,kff,prs,ex, flexi,flexo)
  c *** primary function: to (re)compute the flexibility factor
  c for pipe16, pipe17, pipe18, and pipe60
  c this is accessed by inputting the flexibility factor
  c as any negative number.
  c *** secondary functions: none
  c *** Notice - This file contains ANSYS Confidential information ***
  c *** copyright(c) 2006 SAS IP, Inc. All rights reserved.
  c *** ansys, inc.
  c input arguments:
  c variable (typ,siz,intent)    description
  c etype    (int,sc,in)       - pipe element type (16, 17, 18 or 60)
  c elem     (int,sc,in)       - element number
  c rvrm     (dp,ar(*),in)     - real constants
  c kff      (int,sc,in)       - keyopt for flexibility factor
    (not used for pipe16 or pipe17)
  cprs      (dp,ar(5),in)     - pressures
  c ex       (dp,sc,in)       - young's Modulus
  c flexi    (dp,sc,inout)     - effective in-plane flexibility factor
  c flexo    (dp,sc,inout)     - effective out-of-plane flexibility factor
    (not used for pipe16 or pipe17)
  c output arguments:
  c variable (typ,siz,intent)    description
  c flexi    (dp,sc,inout)     - effective in-plane flexibility factor
  c flexo    (dp,sc,inout)     - effective out-of-plane flexibility factor
    (not used for pipe16 or pipe17)
```
6.3.17. Subroutine UsrShift (Calculates pseudotime time increment)

```
*deck, user
  subroutine UsrShift(dxi,dxihalf,timinc,
    & temp,dtemp,toffst,propsh,nTerms)
  c********************************************************************************
  c calculate pseudotime time increment according
  c to a user specified shift function
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  c input arguments:
  c timinc  (dp,sc,in)         - time increment
  c temp    (dp,sc,in)         - current temperature, t_n+1
  c dtemp   (dp,sc,in)         - temperature increment, t_n+1 - t_n
  c toffst  (dp,sc,in)         - temperature offset to absolute zero
  c           (specified by TOFFST command)
  c propsh  (dp,ar,in)         - Constants for shift function
  c           (User's input using TB,SHIFT,,,USER)
  c nTerms  (int,ar,in)        - number of user defined constants
  c           (specified in TB,SHIFT,,,nTerms,USER)
  c output arguments:
  c dxi      (dp,sc,out)        - pseudotime increment
  c dxihalf  (dp,sc,out)        - pseudotime increment over the upper half span
  c********************************************************************************
```

6.4. Routines for Customizing Material Behavior

This section describes the user routines you use to modify or monitor how materials behave. These routines enable you to perform tasks including:

- Writing your own material constitutive models
- Writing your own plasticity, creep, or swelling laws
- Writing your own hyperelasticity laws
- Updating the nonlinear strain history for a user-defined material
- Checking material data you have defined
- Computing the derivatives of the strain energy density "W" with respect to its invariants
- Controlling hygrothermal growth.

Note

If you write a material behavior routine using any of the ANSYS commands MPDATA, MPDELE, TB, or TBDELE, be aware that these commands interpret the string "_MATL" to mean the current active material when it appears in their MAT field. The "_MATL" is used in conjunction with the library (LIB) option of the MPREAD and MPWRITE commands. MPWRITE inserts "_MATL" in lieu of the specified material number as these commands are written to the material library file. This occurs only when you use the LIB option. When ANSYS reads a material library file written in this format, it interprets "_MATL" to mean the currently active material (as defined by the MAT, MAT command). Do not use the "_MATL" string outside the scope of the MPREAD command.

6.4.1. Subroutine usermat (Writing Your Own Material Models)

This subroutine is applicable to the following elements: LINK180, SHELL181, PLANE182, PLANE183, SOLID185, SOLID186, SOLID187, SOLSH190, BEAM188, BEAM189, SHELL208, and SHELL209.

```
*deck, usermat
  subroutine usermat()
```
& matId, elemId, kDomIntPt, kLayer, kSectPt,
& ldstep, isubst, keycut,
& nDirect, nShear, ncomp, nStatev, nProp,
& Time, dTime, Temp, dTemp,
& stress, ustatev, dsdePl, sedEl, sedPl, epseq,
& Strain, dStrain, epsPl, prop, coords,
& rotateM, defGrad_t, defGrad,
& tsstif, eps2Z2,
& var1, var2, var3, var4, var5,
& var6, var7, var8)

*************************************************************************

*** primary function ***

user defined material constitutive model

Attention:
User must define material constitutive law properly
according to the stress state such as 3D, plain strain
and axisymmetry, plane stress and 3D/1D beam.

a 3D material constitutive model can use for
plain strain and axisymmetry cases.

When using shell elements, a plane stress algorithm
must be use.

gal July, 1999

The following demonstrates a USERMAT subroutine for
a plasticity model, which is the same as TB, BISO,
for different stress states.
See "ANSYS user material subroutine USERMAT" for detailed
description of how to write a USERMAT routine.

This routine calls four routines,
usermat3d.F, usermatps.F usermatbm.F and usermat1d.F, w.r.t.
the corresponding stress states.
Each routine can be also a usermat routine for the specific
element.

*************************************************************************

input arguments
-----------------
matId     (int, sc, i)     material #
elemId    (int, sc, i)     element #
kDomIntPt (int, sc, i)     "k"th domain integration point
kLayer    (int, sc, i)     "k"th layer
kSectPt   (int, sc, i)     "k"th Section point
ldstep    (int, sc, i)     load step number
isubst    (int, sc, i)     substep number
nDirect   (int, sc, in)    # of direct components
nShear    (int, sc, in)    # of shear components
ncomp     (int, sc, in)    nDirect + nShear
nstatev   (int, sc, l)     Number of state variables
nProp     (int, sc, l)     Number of material ocstants
Temp      (dp, sc, in)     temperature at beginning of
time increment
dTemp     (dp, sc, in)     temperature increment
Time      (dp, sc, in)     time at beginning of increment (t)
dTime     (dp, sc, in)     current time increment (dt)
Strain    (dp, ar(ncomp), i) Strain at beginning of time increment
dStrain   (dp, ar(ncomp), i) Strain increment
prop      (dp, ar(nprop), i) Material constants defined by TB,USER
coords    (dp, ar(3), i)   current coordinates
rotateM   (dp, ar(3,3), i) Rotation matrix for finite deformation update
            Used only in 5.6 and 5.7
            Unit matrix in 6.0 and late version
defGrad_t (dp, ar(3,3), i) Deformation gradient at time t
defGrad   (dp, ar(3,3), i) Deformation gradient at time t+dt
input output arguments
--------------
stress (dp,ar(nTesn),io) stress
ustatev (dp,ar(nstatev),io) user state variables
sedEl (dp,sc,io) elastic work
dedPl (dp,sc,io) plastic work
epsseq (dp,sc,io) equivalent plastic strain
tsstif (dp,ar(2),io) transverse shear stiffness
tsstif(1) - Gxz
tsstif(2) - Gyz
tsstif(1) is also used to calculate hourglass stiffness, this value must be defined when low order element, such as 181, 182, 185 with uniform integration is used.
var? (dp,sc,io) not used, they are reserved arguments for further development

output arguments
--------------
keycut (int,sc,io) loading bisect/cut control
0 - no bisect/cut
1 - bisect/cut
(factor will be determined by ANSYS solution control)
dsdePl (dp,ar(ncomp,ncomp),io) material jacobian matrix
epsZZ (dp,sc,o) strain epsZZ for plane stress, define it when accounting for thickness change in shell and plane stress states

*************************************************************************
ncomp 6 for 3D (nShear=3)
ncomp 4 for plane strain or axisymmetric (nShear = 1)
ncomp 3 for plane stress (nShear = 1)
ncomp 3 for 3d beam (nShear = 2)
ncomp 1 for 1D (nShear = 0)

stress and strains, plastic strain vectors
11, 22, 33, 12, 23, 13 for 3D
11, 22, 33, 12 for plane strain or axisymmetry
11, 22, 12 for plane stress
11, 13, 12 for 3d beam
11 for 1D

material jacobian matrix
3D
dsdePl | 1111 1122 1133 1112 1123 1113 |
dsdePl | 2211 2222 2233 2212 2223 2213 |
dsdePl | 3311 3322 3333 3312 3323 3313 |
dsdePl | 1211 1222 1233 1212 1223 1213 |
dsdePl | 2311 2322 2333 2312 2323 2313 |
dsdePl | 1311 1322 1333 1312 1323 1313 |
plane strain or axisymmetric (11, 22, 33, 12)
dsdePl | 1111 1122 1133 1112 1123 1113 |
dsdePl | 2211 2222 2233 2212 2223 2213 |
dsdePl | 3311 3322 3333 3312 3323 3313 |
dsdePl | 1211 1222 1233 1212 1223 1213 |
plane stress (11, 22, 12)
dsdePl | 1111 1122 1112 |
dsdePl | 2211 2222 2212 |
dsdePl | 1211 1222 1212 |
3d beam (11, 13, 12)
dsdePl | 1111 1113 1112 |
dsdePl | 1311 1313 1312 |
dsdePl | 1211 1213 1212 |
1d
dsdePl | 1111 |

*************************************************************************
Subroutine userpl (Writing Your Own Plasticity Laws)

This subroutine is applicable to the following elements: LINK1, PLANE2, LINK8, PIPE20, BEAM23, BEAM24, PLANE42, SHELL43, SOLID45, SHELL51, PIPE60, SOLID62, SOLID65, PLANE82, SHELL91, SOLID92, SHELL93, SOLID95.

```
*deck, userpl
subroutine userpl (elem, intpt, mat, ncomp, kfirst, kfsteq, e, nu, dens,
  x prop, d, ktform, timval, timinc, tem, dtem, toffst, flu, dflu, epel, eppl,
  x statev, usvr, epeq, plwork, sigepl, sigrat, depeq, dt)

*** primary function:  allow users to write their own plasticity laws.
  this logic is accessed with tb,user.
  the below demonstration logic is the same as using
  tb,bkin, without adaptive descent (nropt,,,off).
  Other plasticity rules may require internal
  iterations and/or the more general definition of
  plasticity theory, discussed in the Theory
  Manual.

*** secondary function:  demonstrate the use of user-written plasticity laws
  in this routine:
  a.  update the nonlinear strain history
  b.  compute the material tangent matrix if requested

*** Notice - This file contains ANSYS Confidential information ***

*** ansys(r) copyright(c) 2004
*** ansys, inc.

input arguments:

variable (type, sze, intent)    description

elem    (int, sc, in)    - element number (label)
intpt   (int, sc, in)    - element integration point number
mat     (int, sc, in)    - material reference number
ncomp   (int, sc, in)    - no. of stress/strain components (1,4 or 6)
  1 - x
  4 - x,y,z,xy
  6 - x,y,z,xy,yz,xz
kfirst  (int, sc, in)    - 1 if first time through, 0 otherwise
  (useful for initializing state variables
to a non-zero value)
kfsteq  (int, sc, in)    - 1 if first equilibrium iteration of a
  substep, 0 otherwise
e      (dp, sc, in)    - average elastic modulus
nu      (dp, sc, in)    - average poisson ratio
dens    (dp, sc, in)    - current material density (mass/volume)
prop    (dp, ar(9),in)  - linear material property array
  (ex, ey, ez, gxy, gyz, gxz, nuxy, nuyz, nuxz)
  if ncomp=1 (ex)
d      (dp, ar(ncomp,ncomp),in)- elastic stress-strain matrix
ktform  (int, sc, in)    - request key for tangent matrix formation
  (-1, form tangent .ne.1, do not form)
timval  (dp, sc, in)    - current time value
timinc  (dp, sc, in)    - time increment over this substep
tem     (dp, sc, in)    - temperature at the end of this substep
dtem    (dp, sc, in)    - temperature increment over this substep
toffst  (dp, sc, in)    - temperature offset from absolute zero
flu     (dp, sc, in)    - fluence at the end of this substep
dflu    (dp, sc, in)    - fluence increment over this substep
epel    (dp, ar(ncomp),inout)- modified total strain (trial strain)
  epel = eptot - eppl - eptherm - ...
  if a large strain analysis, epel is
  rotation neutralized and is the hencky
  (i.e. log) strain
eppl    (dp, ar(ncomp),inout)- plastic strain from previous substep
```
6.4.3. Subroutines usercreep and usercr (Defining Viscoelastic/Creep Material Behavior)

ANSYS provides two subroutines to allow you to specify your own creep equations to define the time-dependent viscoelastic/creep behavior of materials. The two subroutines are usercreep and usercr. Use the subroutine usercreep when you issue the TB command with the CREEP option, and with TBOPT = 100. Use the subroutine usercr when you issue the TB command with the CREEP option, with TBOPT = 0, and data constant C6 = 100. Subroutine usercreep is incorporated with an implicit time integration algorithm, while subroutine usercr is incorporated with an explicit time integration algorithm. In general, the implicit time integration algorithm is more effective for long time periods. The explicit creep algorithm is more effective for short time periods such...
as transient analyses with very small time increments, or when the creep behavior of materials is not that significant. In the finite deformation analysis, you should interpret the strain variables in the subroutines as logarithmic strains, and you should interpret the stresses as true stresses.

6.4.3.1. Creep Subroutine usercreep

Use the subroutine usercreep to define viscoplastic/creep behavior of materials when you issue the TB command with the CREEP option, and with TBOPT=100. The subroutine will be called at all integration points of elements for which the material is defined by this command. ANSYS always uses implicit time integration for this creep option. You can use plasticity options (BISO, MISO, NLISO) to define the plastic behavior of materials. Creep and plastic strain will be calculated simultaneously when both creep and plasticity are defined for a material. Through this subroutine, you can specify a "uniaxial" creep law that will be generalized to the multi-axial state by the general time-dependent viscoplastic material formulation implemented in ANSYS. You can use and update internal state variables in the subroutine. The number of state variables has to be defined by TB,STATE.

Please see the TB command description for more information.

*deck, usercreep

SUBROUTINE usercreep (impflg, ldstep, isubst, matId, elemId,
  & kDInPt, kLayer, kSecPt, nstatv, nprop,
  & prop, time, dtime, temp, dtemp,
  & toffst, Ustatev, creqv, pres, seqv,
  & delcr, dcrda)

*****************************************************************************
*** primary function ***
*****************************************************************************

Define creep laws when creep table options are
TB, CREEP with TBOPT=100.
Demonstrate how to implement usercreep subroutine

Creep equation is
  \[ \dot{\text{creq}} := k_0 \times \text{seqv}^n \times \text{creqv}^m \times \exp(-b/T) \]

seqv is equivalent effective stress (Von-Mises stress)
creqv is equivalent effective creep strain
T is the temperature
k0, m, n, b are materials constants,

This model corresponds to primary creep function TBOPT = 1

gal 10.01.1998

*****************************************************************************

--- input arguments ---
Explicit/implicit integration flag (currently not used)
Current load step
Current sub step
number of material index
Element number
Material integration point
Layer number
Section point
Number of state variables
size of mat properties array

This array is passed all the creep constants defined by command
TBDATA associated with TB,CREEP
do not use prop(13), as it is used elsewhere
at temperature temp.
Current time
Current time increment
6.4.3. Subroutines usercreep and usercr (Defining Viscoplastic/Creep Material Behavior)

In contrast to the usercreep subroutine, for the usercr subroutine, you need to specify the creep strain tensor. A detailed explanation of this subroutine follows.

```c
*deck,usercr
user
subroutine usercr (elem,intpt,mat,ncomp,kfirst,kfsteq,e,posn,d,
x proptb,timval,timinc,tem,dtem,toffst,fluen,dfluen,epel,epcrp,
x statev,usvr,delcr)
c
*** primary function:  allow users to write their own creep laws.
c *** secondary function:  demonstrate the use of user-written creep laws
c *** Notice - This file contains ANSYS Confidential information ***
c
*** copyright(c) 2006 SAS IP, Inc.  All rights reserved.
c *** ansys, inc.
c
input arguments:
c variable (type,sze,intent) description
c elem (int,sc,in) - element number (label)
c intpt (int,sc,in) - element integration point number
c mat (int,sc,in) - material reference number
c ncomp (int,sc,in) - no. of stress/strain components (1,4 or 6)
c kfirst (int,sc,in) - 1 if first time through, 0 otherwise
```
Chapter 6: UPF Routines and Functions

6.4.4. Subroutine usersw (Writing Your Own Swelling Laws)

*deck, usersw  
subroutine usersw (option, elem, intpt, mat, proptb, ncomp, epswel,  
x epe, e, nuval, nintp, fluent, dfluent, tem, dtem, toffst, timval, timvnc, usvr)

*** primary function: allow users to write their own swelling laws.
*** secondary function: this logic is accessed with c72 = 10

...
6.4.5. Subroutine UserHyper (Writing Your Own Hyperelasticity Laws)

*deck,UserHyper                              user
subroutine UserHyper(
  \& prophy, incomp, nprophy, invar,
  \& potential, pInvDer)

*******************************************************************************

*** Example of user hyperelastic routine

This example uses Arruda hyperelasticity model
which is the same ANSYS TB,BOYCE

input arguments
--------------
prophy       (dp,ar(*),i)     material property array
nprophy     (int,sc,i)        # of material constants
invar       dp,ar(3)          invariants

output arguments
--------------
incomp      (log,sc,i)         fully incompressible or compressible
potential   dp,sc              value of potential
pInvDer     dp,ar(10)          der of potential wrt i1,i2,j

6.4.6. Subroutine uservp (Updating Nonlinear Strain History for Materials)

*deck,uservp user
  subroutine uservp (elem,mat,kfirst,kfsteq,g,prop,timval,timinc,
   tem,toffst,delepn,svrn,svri,usvr,epeln,epeli,strn,stri,depdt,
   cee,eta,norm)

--- primary function: update the nonlinear strain history for a
user-defined material for the viscoxxx elements
--- secondary functions: compute the material tangent terms
  accessed with tb,user and keyopt(1) = 1

--- copyright(c) 2006 SAS IP, Inc. All rights reserved.
--- ansys, inc.

input arguments:
variable (type,sze,intent) description
elem (int,sc,in) - element number (label)
mat (int,sc,in) - material reference number
kfirst (int,sc,in) - 1 if first time through, 0 otherwise
  (useful for initializing state variables
to a non-zero value)
kfsteq (int,sc,in) - 1 if first equilibrium iteration of a
  substep, 0 otherwise
g (dp,sc,in) - shear modulus
prop (dp,ar(13),in) - linear material property array
  (ex,ey,ez, gxy,gyz,gxz, nuxy,nuyz,nuxz,
   alpx,alpy,alpz, dens)
timval (dp,sc,in) - current time value
timinc (dp,sc,in) - time increment over this substep
tem (dp,sc,in) - temperature at the end of this substep
toffst (dp,sc,in) - temperature offset from absolute zero
delepn (dp,ar(3,3),in) - hencky strain increment over the substep
svrn (dp,sc,in) - state variable from previous substep
svri (dp,sc,inout) - state variable from previous iteration
usvr (dp,ar(*),inout) - additional state variables from previous
  iteration (saved if the nsvr command is
  used)
epeln (dp,ar(3,3),in) - elastic strain from previous substep
epeli (dp,ar(3,3),inout) - elastic strain from previous iteration
strn (dp,ar(3,3),in) - stress from previous substep
stri (dp,ar(3,3),inout) - stress from previous iteration
depdt (dp,sc,inout) - effective inelastic deformation rate
  (d(depl)/dt) from previous iteration

output arguments:
variable (type,sze,intent) description
svri (dp,sc,inout) - updated state variable
usvr (dp,ar(*),inout) - updated additional state variables
epeli (dp,ar(3,3),inout) - updated elastic strain
6.4.7. Subroutine userck (Checking User-Defined Material Data)

```
*deck,userck                     user
  subroutine userck (curmat,ntb,tb)
  c *** primary function:    check the user-defined material data,
  c            input with the TB,user command.
  c *** secondary functions:  none
  c
  c *** copyright(c) 2006 SAS IP, Inc. All rights reserved.
  c *** ansys, inc.
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  c input arguments:
  c curmat   (int,sc,in)       - current material number
  c ntb      (int,sc,in)       - dimension of tb
  c tb       (dp,ar(ntb),in)   - input table
  c
  c output arguments:
  c none
```

6.4.8. Subroutine usermc (Controlling Hygrothermal Growth)

```
*deck,usermc                     user
  subroutine usermc (elem,time,msci,dmsci,mscr,mscra)
  c *** primary function:     user control of hygrothermal growth
  c *** secondary functions:  none
  c presently, called only by shell91
  c
  c in order to activate this user programmable feature,
  c the user must enter the usrcal command.
  c
  c *** copyright(c) 2006 SAS IP, Inc. All rights reserved.
  c *** ansys, inc.
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  c input arguments:
  c elem     (int,sc,in)       - element number
  c time     (dp,sc,in)        - time
  c msci     (dp,sc,in)        - moisture content
  c dmsci    (dp,sc,in)        - change of moisture content
  c mscr     (dp,ar(3),in)     - input material properties for growth
  c
  c output arguments:
  c mscra    (dp,ar(3),out)    - output material properties for growth
```
6.4.9. Subroutine usrfc6 (Defining Custom Failure Criteria)

```plaintext
*deck,usrfc6                              user
    subroutine usrfc6 (elem,matlay,iott,keyer,tem,elim,slim,
     eps, sig, fc6)
```

*deck,usrfc6 (elem,matlay,iott,keyer,tem,elim,slim,
     eps, sig, fc6)

User subroutine for defining your own failure criterion

--- accessed with c6 = -1

This is currently only available with

- Notice - This file contains ANSYS Confidential information

**Input Arguments:**
- `elem` (int,sc,in) - element number
- `elim` (dp,ar(9),in) - failure strains at the current temperature
- `slim` (dp,ar(12),in) - failure stresses and coupling coefficients at the current temperature
- `eps` (dp,ar(6),in) - vector of strains
- `sig` (dp,ar(6),in) - vector of stresses
- `tem` (dp,sc,in) - temperature at this point in the model
- `matlay` (int,sc,in) - material number
- `iott` (int,sc,in) - unit number for writing
- `keyer` (int,sc,inout) - error flag (0 = ok, 1 = error or routine not defined)

**Output Arguments:**
- `keyer` (int,sc,inout) - error flag (0 = ok, 1 = error or routine not defined)
- `fc6` (dp,sc,out) - failure criterion to be processed by solid46, shell191, shell199, or solid191

6.4.10. Subroutines usrfc1 through usrfc5

The source code for subroutines `usrfc1`, `usrfc2`, `usrfc3`, `usrfc4`, and `usrfc5` is identical to subroutine `usrfc6` shown above.

6.4.11. Subroutine UserVisLaw (Defining Viscosity Laws)

```plaintext
*deck,UserVisLaw                              user
    subroutine UserVisLaw
      x (dudx,dudy,dudz,
         dvdx,dvdy,dvdz,
         dwdx,dwdy,dwdz,
         u,v,w,x,y,z,kGeom,
         Vis,Temp,Tref,Pres,Pref,Cf,
         MFrac,DfNSpec,Time,VisNew,toffst)
```

Primary function: To provide a user defined viscosity relationship in terms of the following:
- pressure, temperature, position, time, velocity, & velocity-gradient

This routine is for use with the FLOTRAN elements, Fluid141 and Fluid142 only.

In order to activate this subroutine the user must issue

- FLDA,PROT,VISC,USRV command.

In addition the initial value of viscosity must be specified via

- FLDA,PROP,IVIS,value. This value is not available in this routine.

Optionally the user may specify 4 additional coefficients.
6.4.12. Supporting Function egen

The function egen \((kcomp,ep,nuxy) (function)\) combines \(kcomp\) strain components \((ep)\) per:

\[
\frac{1}{(1 + \text{posn})} \left(- \frac{(ep_2 - ep_1)^2 + (ep_3 - ep_1)^2 + (ep_4 - ep_1)^2}{2} + \frac{-\text{posn}(ep_2 + ep_3 + ep_4)}{2}\right)
\]

typedef, egen

function egen (kcomp,ep,posn)

primary function: combines strain components to give an "overall" strain

used in creep and plasticity calculations

secondary functions: none

formulation of overall value is by:

** Notice - This file contains ANSYS Confidential information **

input arguments:

variable (typ,siz,intent) description

kcomp (int,sc,in) - number of components of strain

ep (dp,ar(6),in) - the strain components

posn (dp,sc,in) - poisson's ratio

output arguments:

variable (typ,siz,intent) description

VisNew (dp,sc,out) - new viscosity
6.5. Routines for Customizing Loads

This section describes the user routines you use to modify or monitor existing ANSYS elements. These routines enable you to perform tasks including:

- Setting custom values for scalar fields (temperatures, heat fluxes, etc.)
- Changing element pressure information
- Changing information about element face convection surfaces
- Changing information about element face heat flux surfaces
- Changing information about element face charge density surfaces
- Changing information about element acceleration/angular velocity.

Activate these routines by issuing the **USRCAL** command or by choosing an equivalent menu path.

### 6.5.1. Subroutine usrefl (Changing Scalar Fields to User-Defined Values)

```
*deck,usrefl
subroutine usrefl(key,iel,ielc,nnod,nodes,time,defalt,nd,dat)

*** primary function: change the scalar fields (temperatures, fluences, heat generation, etc.) to what user desires.
*** secondary functions: none

in order to activate this user programmable feature, the user must enter the usrcal command.
this routine is called at each substep of each load step for which element or nodal temperatures(etc) are used.
it is called for each equilibrium iteration.
the call to get the standard ansys input element or nodal values is made just before entering this routine.

*** copyright(c) 2006 SAS IP, Inc. All rights reserved.
*** ansys, inc.

*** Notice - This file contains ANSYS Confidential information ***

typ=int,dp,log,chr,dcp siz=sc,ar(n) intent=in,out,inout

input arguments:
variable (typ,siz,intent) description
key (int,sc,in) - type of data desired
  = 1 temperatures
  = 2 fluences
  = 3 heat generation rates
  = 4 moisture contents
  = 5 magnetic virtual displacements
iel (int,sc,in) - element number
ielc (int,ar(IELCSZ),in) - array of element type characteristics
nnod (int,sc,in) - number of nodes
nodes (int,ar(nnod),in) - list of nodes
time (dp,sc,in) - time of current substep
defalt (dp,sc,in) - default value (e.g. tunif)
nd (int,sc,in) - size of dat array
dat (dp,ar(nd),inout) - array of data as normally computed by element as selected by key

output arguments:
variable (typ,siz,intent) description
```
6.5.2. Subroutine userpr (Changing Element Pressure Information)

*deck, userpr
user
subroutine userpr (ielc, elem, time, ndat, dat)
*** primary function: change element pressure information.

*** copyright(c) 2006 SAS IP, Inc. All rights reserved.
*** ansys, inc.

in order to activate this user programmable feature,
the user must enter the 'usrcal, userpr' command.

this routine is called at each substep of each load step for which
pressures are used. it is called for each equilibrium iteration.
it is called once per element.
the call to get the standard ansys input pressures is made just before
entering this routine.

input arguments:
variable (typ, siz, intent)    description
ielc (int, ar(IELCSZ), in) - array of element type characteristics
elem (int, sc, in) - element number for operation.
time (dp, sc, in) - time of current substep
ndat (int, sc, in) - number of pressure items for this element
dat (dp, ar(ndat,2), inout) - the element pressure vector
(has input values for each corner
of each face)

output arguments:
variable (typ, siz, intent)    description
dat (dp, ar(ndat,2), inout) - the element pressure vector
(defined input values for each corner
of each face)
dat(1:ndat,1) - real pressures
dat(1:ndat,2) - complex pressures
(surface elements only)

the input array dat may be used in one of three ways:

1. it may be simply passed thru
2. it may be used as a flag(e.g. if dat(1) = -3.0, use
   a certain set of logic)
3. it may be completely ignored and instead defined with new logic

6.5.3. Subroutine usercv (Changing Element Face Convection Surface Information)

*deck, usercv
user
subroutine usercv (elem, ielc, time, nr, u, ndat, hc, tb)
*** primary function: change element face convection surface info

*** copyright(c) 2006 SAS IP, Inc. All rights reserved.
*** ansys, inc.

in order to activate this user programmable feature,
the user must enter the 'usrcal, usercv' command.
the input arguments hc and tb may be used in one of three ways:

1. they may be simply passed thru.
2. they may be used as a flag (e.g. if hc(2) = -3.0, use a certain set of logic).
3. they may be completely ignored.
   and instead redefined with new logic

this routine is called during each substep of each load step.
it is called for each equilibrium iteration.
it is called once per element. it is called only during the heat flow load vector formulation stage, and not during the heat flow evaluation stage.
the call to get the standard ansys input convection surfaces is made just before entering this routine, so this information is available to be modified, if desired.

velocity-dependent film coefficients can be computed by inputting the velocity as the input film coefficient or bulk temperature or by inputting the velocity as a function of location in space. this routine could then compute the effective film coefficient.

input arguments:
variable (typ, siz, intent)    description
elem     (int, sc, in)       - element number for operation.
iec     (int, ar(IELCSZ), in) - array of element type characteristics
time     (dp, sc, in)        - time of current substep
nr       (int, sc, in)       - number of nodal temperatures of the element
u        (dp, ar(nr), in)    - vector of most recent values of the temperatures
ndat     (int, sc, in)       - number of data points per element for example, for solid70, ndat = 24 = 6*4 where 6 = faces per element 4 = corners per face
hc     (dp, ar(ndat), inout) - film coefficients (has input values for each corner of each face)
tb     (dp, ar(ndat), inout) - bulk temperature (has input values for each corner of each face)

output arguments:
variable (typ, siz, intent)    description
hc     (dp, ar(ndat), inout) - film coefficients (defines input values for each corner of each face)
tb     (dp, ar(ndat), inout) - bulk temperature (defines input values for each corner of each face)

6.5.4. Subroutine userfx (Changing Element Face Heat Flux Surface Information)

*deck, userfx
subroutine userfx (ielc, elem, time, nr, u, ndat, dat)
c *** primary function: change element face heat flux surface info
c *** copyright(c) 2006 SAS IP, Inc. All rights reserved.
c *** ansys, inc.
c in order to activate this user programmable feature, the user must enter the 'usrcal, userfx' command.
c this routine is called during each substep of each load step.
it is called for each equilibrium iteration.
it is called once per element. it is called only during the heat flow load vector formulation stage, and not during the heat flow evaluation stage.
the call to get the standard ansys input heat flux surfaces...
6.5.5. Subroutine userch (Changing Element Face Charge Density Surface Information)

*deck, userch
subroutine userch (ielc, ielem, time, nr, u, ndat, dat)
  c *** primary function: change element face charge density surface info
  c  in order to activate this user programmable feature,
  c  the user must enter the usrcal command.
  c  this routine is called during each substep of each load step.
  c  it is called once per element. it is called only during the heat
  c  flow load vector formulation stage, and not during the heat flow
  c  evaluation stage.
  c  the call to get the standard ansys input charge densities of surfaces
  c  is made just before entering this routine, so this information is
  c  available to be modified, if desired.
  c  *** copyright(c) 2006 SAS IP, Inc. All rights reserved.
  c  *** ansys, inc.
  c
  input arguments:
  c  variable (typ,siz,intent) description
  c  ielc  (int,ar(IELCSZ),in) - array of element type characteristics
  c  elem  (int,sc,in) - element number for operation.
  c  time  (dp,sc,in) - time of current substep
  c  nr    (int,sc,in) - number of nodal temperatures
  c  of the element
  c  u     (dp,ar(nr),in) - vector of most recent values of the
  c  temperatures
  c  ndat  (int,sc,in) - number of data points per element
  c  for example, for solid70, ndat = 24 = 6*4
  c  where 6 = faces per element
  c  4 = corners per face
  c  dat   (dp,ar(ndat),inout) - fluxes
  c  (has input values for each corner
  c  of each face)
  
  output arguments:
  c  variable (typ,siz,intent) description
  c  dat   (dp,ar(ndat),inout) - fluxes
  c  (defines input values for each corner
  c  of each face)
  c

the input argument dat may be used in one of three ways:
1. they may be simply passed thru.
2. they may be used as a flag(e.g. if dat(2) = -3.0, use
   a certain set of logic).
3. they may be completely ignored.
   and instead redefined with new logic
6.6. Running ANSYS as a Subroutine

To call the ANSYS program, use the following:

```fortran
program ansys
```

If you are running ANSYS on a UNIX system (but not under Windows), you also can call the ANSYS program as a subroutine in a program you've written. To do so, use the following:

```fortran
subroutine ansys
```

For multiple calls to subroutine ANSYS, you must open and close standard input in the calling routine. (Usually, input and output are FORTRAN units 5 and 6, respectively.) The calling routine can't use the database access routines. But, other user programmable features can use the database access routines freely.

There may be times when ANSYS exits abnormally. Check the `file.err` file to see if ANSYS wrote an exit code to the file before ending. These error codes may help you to understand the problem ANSYS had:

### Table 6.1 ANSYS Exit Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal Exit</td>
<td>14</td>
<td>XOX Error</td>
</tr>
<tr>
<td>1</td>
<td>Stack Error</td>
<td>15</td>
<td>Fatal Error</td>
</tr>
<tr>
<td>2</td>
<td>Stack Error</td>
<td>16</td>
<td>Possible Full Disk</td>
</tr>
<tr>
<td>3</td>
<td>Stack Error</td>
<td>17</td>
<td>Possible Corrupted or Missing File</td>
</tr>
<tr>
<td>4</td>
<td>Stack Error</td>
<td>18</td>
<td>Possible Corrupted DB File</td>
</tr>
<tr>
<td>5</td>
<td>Command Line Argument Error</td>
<td>21</td>
<td>Authorized Code Section Entered</td>
</tr>
<tr>
<td>6</td>
<td>Accounting File Error</td>
<td>25</td>
<td>Unable to Open X11 Server</td>
</tr>
<tr>
<td>7</td>
<td>Auth File Verification Error</td>
<td>30</td>
<td>Quit Signal</td>
</tr>
<tr>
<td>8</td>
<td>Error in ANSYS or End-of-run</td>
<td>31</td>
<td>Failure to Get Signal</td>
</tr>
<tr>
<td>11</td>
<td>User Routine Error</td>
<td>&gt;32</td>
<td>System-dependent Error</td>
</tr>
<tr>
<td>12</td>
<td>Macro STOP Command</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6.7. Defining Your Own Commands

ANSYS, Inc. supplies a set of user routines, named `user01` through `user10`, which you can use to define custom ANSYS commands. To do so, follow these steps:

1. Insert the code for the functions you want to perform into routine `user01` (or `user02`, etc.).
2. Link the routine into the ANSYS program.
3. Issue the ANSYS command `/UCMD` to define a name for a custom command that calls and executes your routine. Use the command format shown below:

```
/UCMD, Cmd, SRNUM
```

**Cmd**

The name for your new command. It can contain any number of characters, but only the first four are significant. The name you specify can not conflict with the name of any ANSYS command or the names of any other commands or macros.
SRNUM
The number of the routine your command should call; that is, a value between 01 and 10. For example, suppose that you create and link in a user routine for a parabolic distribution of pressure, and you name that routine user01. Issuing the command shown below creates a new command, PARB, that when issued calls your parabolic pressure distribution routine:

/UCMD, PARB, 1

To make these "custom command" routines available in all your ANSYS sessions, include the /UCMD commands in your start-up file (START.ANS).

You also can use /UCMD to remove a custom command. To do so, simply use a blank value for Cmd, as shown below:

/UCMD,, 1

This command removes the PARB command. To list all user-defined command names, issue the command /UCMD, STAT.

6.7.1. Function user01

*deck, user01  user
function user01()  
c *** primary function: user routine number 01

*** copyright(c) 2006 SAS IP, Inc. All rights reserved.
c *** ansys, inc.
c *** Notice - This file contains ANSYS Confidential information ***
c /*-------------------------------------------------------------*/
c | this is a user routine that may be used by users to include their |
c | special coding. access to this routine is by the command usr1. the |
c | user may then use this routine to call his/her special routines. |
c | ansys routines to access information in the ansys database may be |
c | found in the "ansys programmer's manual", available from ansys,inc |
c | see user02 for a simple example usage. |
c | routines user03 to user10 are also available. |
C \*******************************************************************/

c input arguments: none

c output arguments:
c    user01   (int, sc, out)      - result code (should be zero)
c                                           (which is ignored for now)
c
C ***********************************************************************
c Functions for accessing data on the command line
C integer function intinfun(iField) - gets an integer from field iField
C double precision function dpinfun(iField) - gets double precision
C character*4 ch4infun(iField) - gets (upper case) 4 characters
C character*8 ch8infun(iField) - gets (mixed case) 8 characters
C character*32 ch32infun(iField) - gets (mixed case) 32 characters
C ***********************************************************************

external wrinqr
integer wrinqr

integer user01, iott

iott = wrinqr(2)

C ***** USER'S CODE IS INSERTED HERE *****
write (iott, 2000)
2000 format (/\***** CALL TO ANSYS, INC DUMMY USER01 *****/*/)
6.7.2. Function user02 (Demonstrates Offsetting Selected Nodes)

*deck, user02            user
function user02()

*** primary function: user routine number 02
--- This demonstration offsets selected nodes with the command:
usr2,dx,dy,dz

*** copyright(c) 2006 SAS IP, Inc. All rights reserved.
*** ansys, inc.

**** Notice - This file contains ANSYS Confidential information ****

<table>
<thead>
<tr>
<th>see user01 for additional information on user routines</th>
</tr>
</thead>
</table>

input arguments: none

output arguments:
user02   (int, sc, out) - result code (should be zero)
(which is ignored for now)

Functions for accessing data on the command line
integer function intinfun(iField) - gets an integer from field iField
double precision function dpinfun(iField) - gets double precision
character*4 ch4infun(iField) - gets (upper case) 4 characters
character*8 ch8infun(iField) - gets (mixed case) 8 characters
character*32 ch32infun(iField) - gets (mixed case) 32 characters

*******************************************************************************

external TrackBegin, TrackEnd
external wrinqr, ndinqr, ndgxyz, ndpxyz, erhandler, dpinfun
integer wrinqr, ndinqr, ndgxyz
double precision dpinfun

integer user02, iott, maxnp, i, ksel
double precision xyz(3), offset(3)

#include "ansysdef.inc"

***** start timing check *****
call TrackBegin ('user02')

maxnp = ndinqr(0, DB_MAXDEFINED)

***** get the desired offsets from the command line *****
offset(1) = dpinfun(1)
offset(2) = dpinfun(2)
offset(3) = dpinfun(4)

do i = 1, maxnp
ksel = ndgxyz(i, xyz(1))
if (ksel .eq. 1) then
xyz(1) = xyz(1) + offset(1)
xyz(2) = xyz(2) + offset(2)
xyz(3) = xyz(3) + offset(3)
call ndpxyz(i, xyz(1))
endif
endo

***** write to output file *****
iott = wrinqr(WR_OUTPUT)
write (iott,2000)
2000 format (/' NODE OFFSET COMPLETE '/)
c      ***** write to GUI window *****
call erhandler ('user02',3000,
x         2,'NODE OFFSET COMPLETE',0.0d0, ' ')
c      ***** required return value *****
user02 = 0
c      ***** end timing check *****
call TrackEnd ('user02')
return
end

6.7.3. Function user03 (Demonstrates Using ANSYS Memory)

*deck, user03
function user03()
c *** primary function: user routine number 03. Gives example of
   ANSYS Memory usage
c
*** copyright(c) 2006 SAS IP, Inc. All rights reserved.
c *** ansys, inc.
c *** Notice - This file contains ANSYS Confidential information ***
c/*******************************************************************/
c| see user01 for additional information on user routines |
c\*******************************************************************/
c
input arguments: none
c
output arguments:
c      user03 (int, sc, out) - result code (should be zero)
c      (which is ignored for now)
c
***********************************************************************
c Functions for accessing data on the command line
c integer function intinfun(iField) - gets an integer from field iField
c double precision function dpinfun(iField) - gets double precision
   character*8 ch8infun(iField) - gets (upper case) 8 characters
   character*32 ch32infun(iField) - gets (mixed case) 32 characters
c***********************************************************************
#include "impcom.inc"
#include "ansysdef.inc"

external TrackBegin, TrackEnd
external wrinqr, ndinqr, ndgxyz, ndnext, fAnsMemAlloc,
x          fAnsMemFree, erhandler, parreturn, parstatus
PTRFTN   fAnsMemAlloc
integer  user03, iott, i, ksel, numnp, node, istat
double precision xyz(3), xmean, ymean, zmean, stdxyz(3),
x          sodx, sody, sodz
c
pointers:
pointer (pdXnodeL,Xnode)
pointer (pdYnodeL,Ynode)
pointer (pdZnodeL,Znode)
double precision Xnode(*), Ynode(*), Znode(*)
c
***** call to start timing *****
call TrackBegin ('user03')
c
Get nodal xyz locations and calculate standard deviation of
x coordinates, y coordinates, & z coordinates
get number of currently selected nodes
numnp = ndinqr(0, DB_NUMSELECTED)

istat = 1
if (numnp .le. 0) go to 999

allocate memory for x, y, & z coordinates of nodes
pdXnodeL = fAnsMemAlloc(numnp, MEM_DOUBLE, 'XCoords ')
pdYnodeL = fAnsMemAlloc(numnp, MEM_DOUBLE, 'YCoords ')
pdZnodeL = fAnsMemAlloc(numnp, MEM_DOUBLE, 'ZCoords ')

loop through all selected nodes
i = 1
node = 0
xmean = 0.0d0
ymean = 0.0d0
zmean = 0.0d0

10 node = ndnext(node)
if (node .gt. 0) then
    get xyz coordinates
    ksel = ndgxyz(node, xyz(1))
    store this node's xyz coordinates
    Xnode(i) = xyz(1)
    Ynode(i) = xyz(2)
    Znode(i) = xyz(3)
    while we're looping, accumulate sums to calculate means
    xmean = xmean + xyz(1)
    ymean = ymean + xyz(2)
    zmean = zmean + xyz(3)
    increment index
    i = i + 1
    loop back up for next selected node
    goto 10
endif

node = 0, at the end of node list

calculate mean of xyz coordinates
xmean = xmean / numnp
ymean = ymean / numnp
zmean = zmean / numnp

calculate standard deviation for xyz coordinates
sodx = 0
sody = 0
sodz = 0
do i = 1, numnp
    sodx = sodx + (Xnode(i) - xmean)**2
    sody = sody + (Ynode(i) - ymean)**2
    sodz = sodz + (Znode(i) - zmean)**2
enddo

stdxyz(1) = sqrt(sodx / (numnp-1))
stdxyz(2) = sqrt(sody / (numnp-1))
stdxyz(3) = sqrt(sodz / (numnp-1))

***** write to output file *****
iott = wrinqr(WR_OUTPUT)
write (iott, 2000) xmean, ymean, zmean,
   x stdxyz(1), stdxyz(2), stdxyz(3)
2000 format (' MEAN FOR X COORDINATES:', G12.5/
   x ' MEAN FOR Y COORDINATES:', G12.5/

6.7.4. Function user04

*deck,user04 user
  function user04()
  c *** primary function: user routine number 04; demonstrates getting a
  c       list of nodes attached to a keypoint, line, or area
  c
  c *** copyright(c) 2006 SAS IP, Inc. All rights reserved.
  c *** ansys, inc.
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  c /* see user01 for additional information on user routines */
  c */
  c
  c input arguments: none
  c
  c output arguments:
  c    user04   (int,sc,out)      - result code (should be zero)
  c     (which is ignored for now)
  c
  c Functions for accessing data on the command line
  c integer function intinfun(iField) - gets an integer from field iField
  c double precision function dpinfun(iField) - gets double precision
  c character*4 ch4infun(iField) - gets (upper case) 4 characters
  c character*8 ch8infun(iField) - gets (mixed case) 8 characters
  c character*32 ch32infun(iField) - gets (mixed case) 32 characters
  c
  c ***********************************************************************
  c
  external wrinqr, ndkpnt, ndline, ndarea, ch4infun, intinfun
  integer wrinqr, ndkpnt, ndline, ndarea, intinfun
  character*4 ch4infun

  integer user04, iott, listk(20), listl(20), lista(20),
  x i, num,ktype, nkpts, nlines, nareas
  character*4 type, lab2

#include "ansysdef.inc"

iott = wrinqr (WR_OUTPUT)

c --- setup with: /UCMD,GNSME,4
c ! gnsme,group,num,type
c ! group = kp, ln, or ar

c ! num = entity number of kp, ln, or ar

c ! type = interior, or all

c
lab2 = ch4infun(2)
write (iott,2010) lab2
2010 format(/' group name (type of entity) = ',a4)

num = intinfun(3)
write (iott,2020) num
2020 format (' entity number =',i4)

if (lab2 .ne. 'KP  ') then
  type = ch4infun(4)
  if (type .eq. 'INTE') then
    write (iott,2030)
2030       format (' interior nodes only ')
    ktype = 0
  elseif (type .eq. 'ALL ') then
    write (iott,2040)
2040       format (' all (interior and edge/end) nodes ')
    ktype = 1
  else
    write (iott,2050)
2050       format ('Only INTE or ALL are acceptable in last field',
               x 'on user-written gnsme command')
  endif
endif

if (lab2 .eq. 'KP  ') then
  nkpnts = ndkpnt (num, listk(1))
  write (iott,2110) nkpnts
2110    format (' number of nodes on keypoint = ',i4)
  write (iott,2115) (listk(i),i=1,nkpnts)
2115    format (' node on keypoint = ',i4)

elseif (lab2 .eq. 'LN  ') then
  nlines = ndline (num,ktype,listl(1))
  write (iott,2120) nlines
2120    format (' number of nodes on line = ',i4)
  write (iott,2125) (listl(i),i=1,nlines)
2125    format (' list of nodes on line'/(3x,i4))

elseif (lab2 .eq. 'AR  ') then
  nareas = ndarea (num,ktype,lista(1))
  write (iott,2130) nareas
2130    format (' number of nodes on area = ',i4)
  write (iott,2135) (lista(i),i=1,nareas)
2135    format (' list of nodes on area'/(3x,i4))

else
  write (iott,2150)
2150    format (' Only KP, LN, or AR are acceptable on user-written ',
           x ' gnsme command')
endif

user04 = 0
return
end
**6.7.5. Functions user05 through user10**

The source code for user routines user05, user06, user07, user08, user09, and user10 is identical to function user01 shown above.

**6.8. Supporting Subroutines**

### 6.8.1. Function GetRForce (Getting Nodal Reaction Force values)

```c
*deck, GetRForce
  function GetRForce (Node, Label, Value)
  c primary function: Get the K * u - F at a node from the rfsum vector.
  c warning: This routine is called after the elements
  c are formed, but before solution. Therefore,
  c F is from the current iteration, but
  c u is from the previous iteration. At convergence,
  c this difference will have little effect.
  c The computations are done immediately after the
  c call to UElMatx.
  c Use the RFSUM command to ask for the summation.
  c Use *GET, Parm, NODE, num, RF, DOFLAB to access the reaction
  c sum from the command line.
  c secondary functions: Return pointer for fast access
  c object/library: usr

  c *** Notice - This file contains ANSYS Confidential information ***
  c Prolog is not CONFIDENTIAL INFORMATION

  c input arguments:
  c variable (typ, siz, intent)    description
  c Node    (int, sc, in)       - Node Number (User)
  c Label    (ch*4, sc, in)      - DOF Label (Upper Case)
  c                                'UX ', 'UY ', 'TEMP', 'VOLT', 'ROTY', etc

  c output arguments:
  c GetRForce (int, func, out)   - status/pointer
  c                           = 0 - data not valid
  c                           > 0 - Rfsum pointer to data for fast access
  c Value    (dp, sc, out)       - Solution value for Node,Label
  c                                All results are in the nodal coordinate
  c                                system

  c example usage:
  c external GetRForce
  c integer GetRForce, ptr, Node2
  c double precision Value
  c #include "handlecom.inc"   (if Value = Rfsum(ptr) form is to be used)
  c      ptr = GetRForce (Node2,'UY ',Value)
  c later...
  c Value = Rfsum(ptr)
```

### 6.8.2. Function GetStackDisp (Getting Current Displacement Values)

```c
*deck, GetStackDisp
  function GetStackDisp (Node, Label, Value)
  c primary function: Get the displacement at a node from the disp vector
  c secondary functions: Return pointer for fast access
  c object/library: usr
```

Programmer's Manual for ANSYS. ANSYS Release 11.0. 002328. © SAS IP, Inc. All rights reserved. Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.
6.8.3. Subroutine ElResultStrt (Getting Load Data from Analysis Results)

*deck,ElResultStrt
  subroutine ElResultStrt (Label,Comp,LabAvg,TypeData,nVal,iLoc)
  c *** primary function:   (post1) Load data for later ElResultGet
  c *** Notice - This file contains ANSYS Confidential information ***
  c (prolog is not confidential)
  c input arguments:
  c Label   (ch*4,sc,in)   - Result Type
  c Comp    (ch*4,sc,in)   - Result Component (8 char for ESTR)
  c LabAvg  (ch*4,sc,in)   - 'AVG ' or 'NOAV' ('AVG ' default)
  c output arguments:
  c TypeData (int,sc,out)  - Code for data type
  c nVal    (int,sc,out)   - Number of values per point
  c iLoc    (int,sc,out)   - Location of Comp in values

6.8.4. Subroutine ElResultGet (Getting Results Values at Selected Points)

*deck,ElResultGet
  subroutine ElResultGet (nPoints,ebest,elcord,TypeData,iLoc, x
  nVal,result)
  c *** primary function:   (post1) Get results at selected points
  c *** Notice - This file contains ANSYS Confidential information ***
  c (prolog is not confidential)
  c input arguments:
  c nPoints  (int,sc,in)   - Number of evaluation points
  c ** from ElInterp ***
  c ebest    (int,ar(nPoints),in)  - Element(s) containing points
  c elcord   (dp,ar(3,nPoints),in) - Element coordinates
  c ** from ElResultStrt ***
6.9. Access at the Beginning and End of Various Operations

You can access the logic just before an ANSYS run begins or just after a run ends, and at many other intermediate points, by using the ANSYS routines listed below. These routines can perform actions such as evaluating results or performing calculations. (None of the routines have input or output arguments.)

Issue the USRCAL command (or use an equivalent menu path) to activate or deactivate these routines.

<table>
<thead>
<tr>
<th>User Subroutine</th>
<th>Is Called</th>
</tr>
</thead>
<tbody>
<tr>
<td>UAnBeg</td>
<td>At ANSYS start-up</td>
</tr>
<tr>
<td>USolBeg</td>
<td>Before solution</td>
</tr>
<tr>
<td>ULDbeg</td>
<td>Before a load step</td>
</tr>
<tr>
<td>USSSbeg</td>
<td>Before a substep</td>
</tr>
<tr>
<td>UitBeg</td>
<td>Before an iteration</td>
</tr>
<tr>
<td>UITFin</td>
<td>After an iteration</td>
</tr>
<tr>
<td>USSFin</td>
<td>After a substep</td>
</tr>
<tr>
<td>ULDFin</td>
<td>After a load step</td>
</tr>
<tr>
<td>USolFin</td>
<td>After solution</td>
</tr>
<tr>
<td>UAnFin</td>
<td>At the end of an ANSYS run</td>
</tr>
</tbody>
</table>

Subroutines USSSbeg, UitBeg, UITFin and USSFin default to reading a command macro file from the current working directory whose name is subroutine.mac (that is, ussfin.mac is read by USSFin.F). No user action to relink the ANSYS program is required for the command macro to be read except that the calling sub-
routine must be activated by the **USRCAL** command. The design of the command reading ability of these sub-
routines is limited to APDL parameter setting commands (**GET**, **SET**, \( a = \text{value} \), etc.) and testing for general ANSYS commands is limited. Commands which are known to work include **DIM**, **STATUS**. Commands which require another line (**MSG**, **VWRITE**) are not allowed. Other commands which are known to not work are the solution loading commands (**D**, **F**, **SFE**, and so on). If these capabilities are required, the user will need to create a Fortran subroutine and link this subroutine into ANSYS, as described in Chapter 5: *Using User Programmable Features (UPFs)*.

While parameter substitution into commands is not permitted, **USSBeg**, and so on were designed to be used in conjunction with dynamic tables and parameter substitution from the user subroutine. As an example, consider an ANSYS table defined as \( d_5 = f(\text{par1}) \). If \( d_5 \) contains values of displacement as a function of \( \text{PAR1} \), then \( d_5 \) may be used as a constraint, as

\[
*\text{dim}, d_5, \text{table}, 10, 1, 1, \text{PAR1} \\
d_5(1)=0, .1, .25, \\
/\text{solu} \\
d_5, \text{ux}, \%d_5\%
\]

Modify the value of \( \text{PAR1} \) in **USSBeg.MAC** and the constraint on node 5, \( \text{ux} \) can then be modified in the middle of a load step.

The following is an example of a valid input that may be read by **USSBeg**, **UItBeg**, **UItFin** and **USSFin**.

```plaintext
/COM, SAMPLE ussfin.mac 
a=5
b=mx(1)                           ! *get function is ok
*get,c,active,solu,Time,cpu       ! *get is ok
*dim,array,,6                     ! array parameters are ok
array(1) = 1
array(2) = 2
array(3) = 3
array(4) = 4
array(5) = 5
array(6) = 6
*vlen,g,3                        ! vector operations are ok
*vfun, array(4), copy, array(1) 
*stat 
*stat, array(1) 
array(1)=
nnode = ndinqr(0,14)
*dim, array,,nnode 
*vget, array(1), NODE, 1, NSEL 
*stat, array(1) 
array(1)=
/eof 

/COM, COMMANDS BELOW THIS LINE ARE KNOWN TO NOT WORK
p,1,6,2000                       ! commands DO NOT work
d,1,uy,.1                       
*msg, note
THIS IS A TEST MESSAGE
*vwrite, array(1) 
(/ b = ,f10.4)
```

### 6.10. Creating Your Own Optimization Routine

One way to customize the ANSYS program for your site’s specific needs is to substitute your own external optimization for the standard ANSYS optimization algorithm. You can do so using either of these methods:

- Link a user routine within the ANSYS program.
- Run a stand-alone optimization program using the ANSYS optimization "save" file, Jobname.OPT.
This chapter describes both methods. You can find additional information on design optimization techniques and procedures in the Advanced Analysis Techniques Guide.

6.10.1. Linking a Custom Optimization Program to ANSYS

If you are performing iterative analyses for design optimization, and you wish to use the standard ANSYS optimizer, you simply choose GUI path Main Menu>Design Opt or issue the ANSYS command /OPT.

However, if you plan to use an optimization routine of your own design, you must do the following:

1. Define parameters for your external optimization routine, using either the OPUSER command or GUI path Main Menu>Design Opt>Method/Tool. (For more information about OPUSER, see the Commands Reference.)

2. Either issue the OPTYPE,USER command or choose menu path Main Menu>Design Opt>Method/Tool and specify User Optimizer.

3. Issue the OPEXE command or choose GUI path Main Menu>Design Opt>Run.

Issuing the OPTYPE,USER command (or its GUI equivalent) instructs the ANSYS program to bypass the standard ANSYS design optimizer logic and instead execute your custom optimization routine.

The userop routine (below) includes a flow chart showing how a user-defined optimization routine interacts with the ANSYS program. It also contains a commented example that shows you sample ANSYS command input for design optimization, done with a user-customized version.

For information about the kopusr variable and the userop routine, see the next section.

6.10.2. Subroutine userop (Defining a Custom Optimization Routine)

Instead of creating your custom design optimization routine from scratch, you can simply tailor the userop routine available on your ANSYS distribution medium. Defined within this routine are a number of variables to which you assign the values you want.

Below is a listing of the header information for userop which includes the inputs and outputs and an example user optimization:
optvar = optimization variables (see documentation in cmopt.inc)

*** simplified flowchart of how user optimization interfaces with ansys

***************
*               *
* start         *
*               *
***************

i
i
i
vvv
v

***************
*               *
* /opt module   *
* opt commands  *
* OPEXE
*               *
***************
i
i
i
i
vvv
v

***************
*               *
* ansys analysis*
* use setup file*
*               *
***************

i
i
i
i
vvv
v

***************
*               *
* yes           *
* kopusr > 0    *
***************

i
i
i
i
i
i
i
i
vvv
v

***************
*               *
*               *
***************


When you finish customizing the userop routine, you relink it using either of the methods described in the Installation and Configuration Guide. You must include the cmopt common block in your routine, to allow it to pass common data (such as design variables, state variables, and objective function values) to and from the ANSYS program.

### 6.10.3. Structuring Your Input

When your userop optimization routine finishes executing, program control returns to the ANSYS design optimizer, so that the ANSYS commands on Jobname.LOOP can execute for the next optimization loop. To use your own customization routine, you should issue the following ANSYS commands (in the order shown):

---

**Example Input for User Optimization**

```plaintext
*** sample input for user optimization ***

```c
! batch,list       ! use batch mode
x1=5               ! initial value for dv x1
x2=5               ! initial value for dv x2
/prep7             ! enter prep
y = x1*x1 + x2*x2  ! define function y
g = x1 - x2       ! define function g
/finish            ! finish prep
/opt               ! enter opt
opvar,x1,dv,1,10   ! define 1st dv (range 1 to 10)
opvar,x2,dv,1,10   ! define 2nd dv (range 1 to 10)
opvar,y,obj,,1     ! define objective function (tol = 1)
opvar,q,sy,1       ! define state variable (lower limit 1)
/execute           ! execute opt
/olist,all         ! list results
/finish
```

---

When you finish customizing the userop routine, you relink it using either of the methods described in the Installation and Configuration Guide. You must include the cmopt common block in your routine, to allow it to pass common data (such as design variables, state variables, and objective function values) to and from the ANSYS program.
6.11. Memory Management Routines

ANSYS provides UPF routines you can use for memory management.

6.11.1. Using the Memory Management Routines

ANSYS uses a dynamic memory manager that overlays the system `malloc` and `free` functions and provides a mechanism for accessing the memory from FORTRAN as well as c and c++. Since the UPF routines are provided in FORTRAN, we will be discussing the FORTRAN access routines.

You may certainly use the system `malloc` and `free` functions or, for FORTRAN, the `allocate` system function. However, you may end up competing with ANSYS for memory, and for large problems there may be insufficient system memory to perform the function.

Dynamic memory is done through Cray-style pointers, where a dynamically allocated array is defined via the construct

```fortran
pointer (piArray,Array)
integer Array(*)
```

and memory space for the array is allocated by assigning the pointer, in this case `piArray`, to the allocated memory space:

```fortran
piArray = fAnsMemAlloc (size,...)
```

To use the ANSYS memory manager in a UPF, follow these steps:

1. Define the dynamically allocated arrays:
   ```fortran
   pointer (piArray,Array), (pdData,Data)
   integer Array(*)
   double precision Data(*)
   ```

2. Initialize the pointers as follows:
   ```fortran
   piArray = PTRFTNNULL
   pdData = PTRFTNNULL
   ```

3. Allocate space for an array or arrays, as follows:
   - For integer numbers:
     ```fortran
     piArray = fAnsMemAlloc(ileng,MEM_INTEGER,C16Label)
     ```
   - For double precision numbers:
     ```fortran
     pdArray = fAnsMemAlloc(dleng,MEM_DOUBLE,C16Label)
     ```
   - For complex numbers:
     ```fortran
     pcArray = fAnsMemAlloc(cleng,MEM_COMPLEX,C16Label)
     ```
   - For real numbers:
prArray = fAnsMemAlloc(rleng,MEM_REAL,C16Label)

Where the arguments are:

- xlen is the desired size of the array
- MEM_xxx is the keyword indicating the type of data
- C16Label is a character*16 name of the memory block

You must include the ansysdef.inc include file to get the parameter values of MEM_INTEGER, MEM_DOUBLE, MEM_COMPLEX, and MEM_REAL.

**Note**

If there is insufficient memory, fAnsMemAlloc returns “PTRFTNNULL”.

4. Use the arrays.
5. Deallocate the space using the fAnsMemFree subroutine, as follows:

    call fAnsMemFree (piArray)

The next two sections provide input and output listings for the memory management routines.

For an example, see Section 6.7.3: Function user03 (Demonstrates Using ANSYS Memory), which appears earlier in this chapter.

### 6.11.2. Function fAnsMemAlloc (Allocating Space and Returning a Pointer)

*deck,fAnsMemAlloc

    function fAnsMemAlloc (iLen, key, c16Label)

    c primary function: Get A Block of Space from mem manager and Return Pointer

    c keywords: integer function for mem allocate

    c object/library: mem

    c *** Notice - This file contains ANSYS Confidential information ***

    c input arguments:
    c     iLen (int,sc,in)       - length of the block (in data elements)
    c     c16Label (chr*16,sc,in)  - 16 character name for the Block
    c     key (int,sc,in)        - type of data for this block (see ansysdef)

    c output arguments:
    c     fAnsMemAlloc (PTRFTN,sc,out) - Pointer to this data block -- needs to be
    c                     tied to a local variable in the calling
    c                     routine

### 6.11.3. Subroutine fAnsMemFree (Deallocating Space)

*deck,fAnsMemFree

    subroutine fAnsMemFree (memPtr)

    c primary function: Free a Data Block, given a pointer

    c keywords: subroutine to free a mem block

    c object/library: mem

    c *** Notice - This file contains ANSYS Confidential information ***
6.12. Parameter Processing Routines

The ANSYS distribution medium contains three routines you can use for parameter processing: \texttt{pardim}, \texttt{parevl}, and \texttt{pardef}.

6.12.1. Subroutine \texttt{pardim} (Creating a Dimensioned Parameter)

\begin{verbatim}
*deck,pardim
    subroutine pardim (cName,labl4,nDim,nxyz,cLabels)
    c *** primary function: create a dimensioned parameter
    c *dim,parm32,type,d1,d2,d3,cName1,cName2,cName3
    c *dim,parm32,type,d1,cName1
    c *dim,parm32,type,d1,d2,d3,d4,d5,cName1,cName2,cName3,cName4,cName5
    c *** Notice - This file contains ANSYS Confidential information ***
    c input arguments:
    c    cName    (chr*32,sc,in)    - the name of the parameter to create
    c    labl4    (chr*4,sc,in)     - 'TABL' or 'ARRA' or 'CHAR' or 'STRI'
    c    nDim     (int,sc,in)       - Dimension of array
    c    nxyz     (int,ar(nDim),in) - the dimensions of the array
    c    cLabels  (chr*32,ar(nDim),in) - Names for the directions in table
    c output arguments:  none
\end{verbatim}

6.12.2. Function \texttt{parevl} (Finding and Evaluating a Parameter)

\begin{verbatim}
*deck,parevl
    subroutine parevl (ParName,nDim,subc,lvl,dpValue,chValue,kerr)
    c *** primary function: find and evaluate a parameter
    c *** Notice - This file contains ANSYS Confidential information ***
    c input arguments:
    c    ParName  (chr*(PARMSIZE),sc,in) - the name of the parameter
    c                                       (must be upper case, left justified)
    c    nDim     (int,sc,in)            - the number of subscripts (0,scaler)
    c    subc     (dp,ar(*),in)          - values for the subscripts (if any)
    c    lvl      (int,sc,in)            - 0,1  no error output  2, report error
    c                                       -1, set kerr flag with no anserr call
    c output arguments:
    c    dpValue (dp,sc,out)            - the value of the parameter (may be a
    c                                   packed character*8
    c    chValue (ch*128,sc,out)        - character output
    c    kerr    (int,sc,out)           - error flag (0,ok -1,output is packed
    c                                       0-ok, 1-error, 2-error but TINY is used
    c                                       -2, output is string in chValue
\end{verbatim}

6.12.3. Subroutine \texttt{pardef} (Adding a Parameter)

\begin{verbatim}
*deck,pardef
    subroutine pardef (label,ctype,nval,subc,valuein,kerr,string)
    c *** primary function: add a parameter to parameter list
    c *** Notice - This file contains ANSYS Confidential information ***
\end{verbatim}
input arguments:

- **label** (chr*(PARMSIZE), sc, in) - name of parameter
  - label is a character variable that
  - contains the name of the parameter that
  - is to be defined. (Length = PARMSIZE characters)

- **ctype** (int, sc, in) - 0, dp 1, character 2, string
  - ctype is an integer key which describes
  - the type of data that the parameter data
  - holds. This would also indicate the
  - contents of "value" (arg 5).
  - 0=double precision data
  - 1=character data packed in value
  - 2=character data in string

- **nval** (int, sc, in) - number of subscripts
  - nval is the number of subscripts that the
  - "label" (arg 1) contains.
  - 1=single dimensioned variable (ex. x(10))
  - 2=double dimensioned variable (ex. y(10,3))
  - 3=triple dimensioned variable (ex. z(10,3,2))
  - -1=delete this parameter from the internal
  - tables.

- **subc** (dp, ar(*), in) - values of subscripts
  - subc is a double precision vector that
  - contains the subscripts of "label" (arg 1).
  - There should be enough values defined to
  - match "nval" (arg 3). For example if "x"
  - was dimensioned as "x(10,3,2)" and you wanted
  - to set "x(5,1,1)=123.0", then "nval" (arg 3)
  - should be set to 3, and "subc" should be set
  - to 5.0, 1.0, 1.0, and "value" (arg 5) should
  - be 123.0. Another example is if "y" was
  - dimensioned to as "y(20,20)" and you were
  - setting "y(5,8)=987", then "nval" (arg 3) should
  - be set to 2 and "subc" should be set to 5.0,
  - 8.0, 0.0, and "value" (arg 5) should be 987.0.

- **valuein** (dp, sc, in) - the value for this parameter
  - (should be a packed character*8 if
  - ctype=1. To pack a char into a dp
  - variable use "call chtodp(ch8,dp)").
  - To unpack a dp variable into a char
  - use "call dptoch(dp,ch8)"
  - Value is the data value that is to be stored for
  - "label" (arg 1). If "ctype=1" (arg 2) then this
  - value would be a "packed character" data from the
  - "chtodp" Ansys function.

output arguments:

- **kerr** (int, sc, out) - error flag (0-ok, 1-error)
  - kerr is an integer error flag that is
  - returned to the calling subroutine. Any
  - non zero number would indicate an error
  - was detected in subroutine "pardef"
6.13. Miscellaneous Useful Functions

The ANSYS program has several miscellaneous functions you may find useful for working with UPFs:

- The **erhandler** routine displays output messages (notes, warnings, and errors).
- The **RunCommand** function lets you issue an ANSYS command from within a user routine.
- The **GetStackDisp** routine retrieves current displacement values.
- The **/UNDO** command writes an "undo" file at critical points as a user routine executes.
- The **/HOLD** command allows you to synchronize multiple tasks in ANSYS.
- The **/TRACK** command enables you to do program tracing and timing.

For further descriptions of **erhandler** and **/TRACK**, see Chapter 8: Subroutines for Users' Convenience. For details about the **GetStackDisp** function, see Section 6.8.2: Function GetStackDisp (Getting Current Displacement Values).

6.13.1. Using Function RunCommand

This function enables you to execute an ANSYS command from within a user routine. Inputs and outputs for **RunCommand** are as follows:

```
*deck,RunCommand
function RunCommand (nChar,command)
```

- **c primary function:** Execute an ansys command
- **c object/library:** upf

```
c *** Notice - This file contains ANSYS Confidential information ***
```

- **c input arguments:**
  - **nChar** (int,sc,in) - Length of the command string (8 min)
  - **command** (ch*(nChar),sc,in) - A character string containing a valid ANSYS command

- **c output arguments:**
  - **RunCommand** (int,sc,out) - An internally defined value, ignore

6.13.2. Using the **/UNDO** Command

The "undo" file you create by issuing the **/UNDO** command is similar to the **File.DB** file created when you issue the **SAVE** command. The **/UNDO** command format is:

```
/UNDO, Action
```

- **Action**
  - **ON**, to write the undo file
  - **OFF**, to prevent the undo file from being written
  - **PROMPT**, to have ANSYS ask permission before writing the file
  - **STATUS**, to restore the file as it existed after executing the last command issued before the **/UNDO** command.

6.13.3. Using the **/HOLD** command

Issue the **/HOLD** command to synchronize tasks in ANSYS. The ANSYS program can synchronize tasks at the end of each results file set.

```
/HOLD, Filename, TimeInterval, Timeout
```
Filename
The eight-character name of a message file. If the named file exists, the ANSYS program reads a command from the file and then deletes the file.

TimeInterval
The length of time, in seconds, that ANSYS waits before trying to read the message file again.

Timeout
The maximum length of time, in seconds, that ANSYS can wait between attempts to read the file.
Chapter 7: Accessing the ANSYS Database

This chapter describes how you can retrieve information in the ANSYS database (or store information in the database) by linking subroutines you create into the ANSYS program.

You can use the database access routines with any of the user-programmable features. For example, you can create your own ANSYS commands and use them to execute database access routines (or have a database access routine call a user-defined command).

7.1. Inputs and Outputs for Database Access Routines

The descriptions of the database access routines or functions within this chapter describe both the input arguments and output arguments. Argument information includes the argument’s type, size and intent.

- Argument type is one of the following:
  - int - integer
  - dp - double precision
  - log - logical
  - chr - character
  - dcp - double precision complex

- Argument size is one of the following:
  - sc - scalar variable
  - ar(n) - array variable of length n
  - func - functional return value

- Argument intent is one of the following:
  - in - input argument
  - out - output argument
  - inout - both an input and an output argument

7.2. Types of Database Access Routines

The rest of this chapter describes the functions and subroutines available for accessing information in the ANSYS database. The function and subroutine descriptions are grouped into sets: nodal information functions, element attribute information functions, and so on.

7.3. Routines for Selecting and Retrieving Nodes and Elements

7.3.1. ndnext Function (Getting the Next Node Number)

*deck, ndnext

function ndnext (next)

c *** primary function: get the number of the next selected node

c *** Notice - This file contains ANSYS Confidential information ***

c input arguments:
next (int, sc, in) - the last node number used
= 0 - use for initial value

c output arguments:
ndnext (int, func, out) - the next selected node number
= 0 - no more nodes
7.3.2. ndprev Function (Getting the Number of the Previous Selected Node)

```c
*deck,ndprev
function ndprev (next)
c *** primary function: get the number of the previous selected node
c *** Notice - This file contains ANSYS Confidential information ***
typ=int,dp,log,chr,dcp siz=sc,ar(n),func intent=in,out,inout

input arguments:
next (int,sc,in) - the next node number used
  = 0 - use for initial value

output arguments:
ndprev (int,func,out) - the previous selected node number
  = 0 - no more nodes
```

7.3.3. ndnxdf Function (Getting the Number of the Next Defined Node)

```c
*deck,ndnxdf
function ndnxdf (next)
c *** primary function: get the number of the next defined node
c *** Notice - This file contains ANSYS Confidential information ***
typ=int,dp,log,chr,dcp siz=sc,ar(n),func intent=in,out,inout

input arguments:
next (int,sc,in) - the last node number used
  = 0 - use for initial value

output arguments:
ndnxdf (int,func,out) - the next defined node number
  = 0 - no more nodes
```

7.3.4. ndsel Function (Selecting, Unselecting, Deleting, or Inverting a Node)

```c
*deck,ndsel
subroutine ndsel (ndmi,ksel)
c *** primary function: to select, unselect, delete, or invert a node.
c *** secondary functions: none.
c *** Notice - This file contains ANSYS Confidential information ***
typ=int,dp,log,chr,dcp siz=sc,ar(n),func intent=in,out,inout

input arguments:
ndmi (int,sc,in) - node number
  = 0 - all nodes
  < 0 - do not delete CPs and CEQNs
  (merge/offset/compress)
ksel (int,sc,in) - type of operation to be performed.
  ksel = 0 - delete node.
  = 1 - select node.
  =-1 - unselect node.
  = 2 - invert select status of node.

output arguments:
none.
```
7.3.5. elnext Function (Getting the Number of the Next Element)

*deck,elnext
  function elnext (next)
  c *** primary function: get the number of the next selected element
  c *** Notice - This file contains ANSYS Confidential information ***
  c    input arguments:
  c     next    (int,sc,in)       - the last element number used
  c     = 0 - use for initial value
  c    output arguments:
  c     elnext   (int,func,out)  - the next selected element
  c     = 0 - no more elements

7.3.6. elprev Function (Getting the Number of the Previous Selected Element)

*deck,elprev
  function elprev (prev)
  c *** primary function: get the number of the previous selected element
  c *** Notice - This file contains ANSYS Confidential information ***
  c    typ=int,dp,log,chr,dcp   siz=sc,ar(n),func    intent=in,out,inout
  c    input arguments:
  c     prev     (int,sc,in)       - the last element used
  c     = 0 - use for initial value
  c    output arguments:
  c     elprev   (int,func,out)  - the previous selected element
  c     = 0 - no more elements

7.3.7. elnxdf Function (Getting the Number of the Next Defined Element)

*deck,elnxdf
  function elnxdf (next)
  c *** primary function: get the number of the next defined element
  c *** Notice - This file contains ANSYS Confidential information ***
  c    typ=int,dp,log,chr,dcp   siz=sc,ar(n),func    intent=in,out,inout
  c    input arguments:
  c     next     (int,sc,in)       - the last element used
  c     = 0 - use for initial value
  c    output arguments:
  c     elnxdf   (int,func,out)  - the next defined element
  c     = 0 - no more elements

7.3.8. elsel Subroutine (Selecting, Unselecting, Deleting, or Inverting an Element)

*deck,elsel
  subroutine elsel (ielel,ksel)
  c *** primary function: to select, unselect, delete, or invert an element.
  c *** Notice - This file contains ANSYS Confidential information ***
  c    input arguments:
7.4. Node Information Routines

7.4.1. ndinqr Function (Getting Information About a Node)

The primary function of ndinqr is getting information about a node. You can also use this function to set the current node pointer to this node.

```
*deck,ndinqr
function ndinqr (node,key)
  c *** primary function:    get information about a node.
  c *** secondary functions: set current node pointer to this node.
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c    node     (int,sc,in)   - node number
  c                                   Should be 0 for key=11, DB_NUMDEFINED,
  c                                   DB_NUMSELECTED, DB_MAXDEFINED, and
  c                                   DB_MAXRECLENG
  c    key      (dp,sc,in)     - key as to information needed about
  c                                   the node.
  c      = DB_SELECTED  - return select status:
  c        ndinqr = 0 - node is undefined.
  c        =-1 - node is unselected.
  c        = 1 - node is selected.
  c      = DB_NUMDEFINED  - return number of defined nodes
  c      = DB_NUMSELECTED  - return number of selected nodes
  c      = DB_MAXDEFINED  - return highest node number defined
  c      = DB_MAXRECLENG  - return maximum record length (dp words)
  c      =  2, return length (dp words)
  c      =  3,
  c      =  4, pointer to first data word
  c      = 11, return void percent (integer)
  c      = 17, pointer to start of index
  c      =-1,
  c      =-2, superelement flag
  c      =-3, master dof bit pattern
  c      =-4, active dof bit pattern
  c      =-5, solid model attachment
  c      =-6, pack nodal line parametric value
  c      =-7, constraint bit pattern
  c      =-8, force bit pattern
  c      =-9, body force bit pattern
  c      =-10, internal node flag
  c      =-11, orientation node flag =1 is =0 is not
  c      =-11, contact node flag =0
  c      =-12, constraint bit pattern (for DSYM)
  c      =-13, if dof constraint written to file.k (for LSDYNA only)
  c      =-14, nodal coordinate system number (set by NROTATE)
  c      =-101, pointer to node data record
  c      =-102, pointer to angle record
  c      =-103,
  c      =-104, pointer to attached couplings
  c      =-105, pointer to attached constraint equations
  c      =-106, pointer to nodal stresses
  c      =-107, pointer to specified disp's
  c      =-108, pointer to specified forces
  c      =-109, pointer to x/y/z record
```

Programmer’s Manual for ANSYS . ANSYS Release 11.0 . 002328 . © SASIP, Inc. All rights reserved. 
Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.
7.4.2. getnod Function (Getting a Nodal Point)

*deck, getnod
  subroutine getnod (node,v,kerr,kcrot)
  c *** primary function: get a nodal point
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c  node (int,sc,in) - node number
  c  kerr (int,sc,inout) - message flag
  c    = 0 - print no message if node is unselected
  c    or undefined
  c    = 1 - print message if node is undefined
  c    or unselected
  c  kcrot (int,sc,in) - output coordinates in this coordinate system.
  c    if kcrot is negative, output theta and
  c    phi coordinates in radians
  c output arguments:
  c  v (dp,ar(6),out) - Coordinates (first 3 values) and rotation
  c    angles (last 3 values)
  c  kerr (int,sc,inout) - select status
  c    = 0 - node is selected
  c    = 1 - node is not defined
  c    =-1 - node is unselected

7.4.3. putnod Function (Storing a Node)

*deck, putnod
  subroutine putnod (node,vctn,kcrot)
  c *** primary function: store a node
  c *** secondary functions: display node if in immediate mode.
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c  node (int,sc,in) - node number
  c  vctn (dp,ar(6),in) - array of 3 nodal coordinates and
  c    3 nodal rotation angles.
  c  kcrot (int,sc,in) - local coordinate system in which the nodal
  c    coordinates and angles are defined
  c output arguments: none.

7.4.4. ndgall Function (Getting the XYZ/Rotation Coordinates Vector for a Node)

*deck, ndgall
  function ndgall (node,xyz)
  c *** primary function: get x,y,z,rotx,roty,rotz vector for a node.
  c *** Notice - This file contains ANSYS Confidential information ***
7.4.5. ndspgt Subroutine (Getting the Nodal Solution for a Node of an Element)

```c
    *deck,ndspgt
    subroutine ndspgt (node,dofs,ndof,nrot,xyzang,nuvect,unode)
    c *** primary function: get the nodal solution for a node of an element
    c *** Notice - This file contains ANSYS Confidential information ***
    c input arguments:
    c      node     (int,sc,in)    - The node number
    c      dofs     (int,ar(DOFBITLENG),in) - The dofs to retrieve for the node.
    c          dof = degree of freedom
    c          The dofs array should be zeroed out, except for the needed parts.
    c          dofs is a bit pattern with true bits
    c          representing the GLOBAL Dof set desired.
    c          That is, dofs(1) is used for UX to SP06,
    c          and dofs(2) is used for TBOT to TTOP.
    c          See ECHPRM for details. For example,
    c          dofs(1) = UX + TEMP
    c          dofs(2) = TE3
    c          TTOP is a special case. If you want
    c          TTOP alone, use:
    c          dofs(2) = ibset(0,TTOP)
    c          If TBOT and TTOP are desired, you must use:
    c          dofs(2) = TBOT
    c          dofs(2) = ibset(dofs(2),TTOP)
    c      ndof     (int,sc,in)    - The number of node dofs (1, 2 or 3).
    c      nrot     (int,sc,in)    - Key to rotate dofs from nodal to global
    c          coordinate systems.
    c          if 0, none. if 2, 2-d. if 3, 3-d
    c          if > 0, dof set must include and only
    c          include all terms of the vector (e.g.
    c          UX,UY,UZ, or AX,AY,AZ).
    c      xyzang   (dp,ar(6),in)  - The xyz virgin node coordinates
    c          (including angles). Not used if
    c          nrot = 0 or ndof < 2.
    c      nuvect   (int,sc,in)    - Number of vectors to retrieve. Can vary
    c          between 1 and 5. Normally 1 is what is
    c          wanted. Other vectors include previous
    c          values and/or velocities. See elucom for
    c          all possibilities. Contents are analysis
    c          type dependent.
    c output arguments:
    c      unode    (dp,ar(ndof,nuvect),out) - Element nodal solution vectors in
    c          the global coordinate system.
```

7.5. Element Attribute Routines

7.5.1. elmiqr Function (Getting Information About an Element)

```c
    *deck,elmiqr
    function elmiqr (ielem,key)
    c *** primary function: get information about an element.
```
7.5.2. elmget Function (Getting an Element's Attributes and Nodes)

*deck,elmget
    function elmget (ielem,elmdat,nodes)
    c *** primary function: get element attributes and nodes.
    c *** Notice - This file contains ANSYS Confidential information ***
    c
    c

7.5.2. elmget Function (Getting an Element's Attributes and Nodes)
7.5.3. elmput Subroutine (Storing an Element)

```
*deck,elmput

subroutine elmput (ielem,elmdat,nnod,nodes)
     c *** primary function:    store element attributes and node numbers.
     c *** secondary functions: set current element pointer to this element.
     c *** Notice - This file contains ANSYS Confidential information ***
     c *** NOTICE - The user is also responsible for defining the centroid for the
     c     element using the elmpct subroutine. Calling the elmput
     c     subroutine will NULL the element centroid previously defined.
     c
     c     input arguments:
     c        ielem    (int,sc,in)       - element number
     c        elmdat   (int,ar(EL_DIM),in) - element attributes.
     c        elmdat(EL_MAT)  - material number
     c     (EL_TYPE)  - element type
     c        (EL_REAL)  - real constant number
     c        (EL_SECT)  - section number
     c        (EL_CSYS)  - coordinate system number
     c     (EL_DEAD)  - death flag (bit 0)
     c     if clear - alive
     c     if set   - dead
     c        (EL_SOLID) - solid model reference
     c        (EL_SHAPE) - 100*shape + specific shape
     c        (EL_OBJOPTIONS) - reserved
     c     (EL_PEXCLUDE) - p element include flag
     c     (bit 0)
     c     if clear - include
     c     (element may need to have its p-level increased)
     c     if set   - exclude
     c     (element does not need to have its p-level increased)
     c     EL_PEXCLUDE is also used for the LSDYNA part number (trh 9/05)
     c
     c     nodes (int,ar(*),out) - node numbers for element.
     c
```

7.5.4. etyiqr Function (Getting a Data Item About an Element Type)

```
*deck,etyiqr

function etyiqr (itype,key)
     c *** primary function:    get information about an element type.
     c *** Notice - This file contains ANSYS Confidential information ***
```
7.5.5. etyget Function (Getting Information About an Element Type)

*deck,etyget
   function etyget (itype,ielx)
   c *** primary function: get element type data.

   c *** Notice - This file contains ANSYS Confidential information ***

   c     input arguments:
   c        itype    (int,sc,in)       - element type number

   c     output arguments:
   c        etyget   (int,func,out)    - status of element type.
   c                                      = 0 - element type is undefined.
   c                                      < 0 - number of data items on unselected
   c                                      element type.
   c                                      > 0 - number of data items on selected
   c                                      element type.
   c        ielx     (int,ar(*),out)   - element type data. see elccmt for
   c                                      description of data.

7.5.6. etyput Subroutine (Storing Element Type Data)

*deck,etyput
   subroutine etyput (itype,n,ielx)
   c *** primary function: store element type data.

   c *** Notice - This file contains ANSYS Confidential information ***

   c     input arguments:
   c        itype    (int,sc,in)       - element type number for operation.
   c        n        (int,sc,in)       - length of data vector to store.
   c        ielx     (int,ar(*),in)    - element type data. see elccmt for
   c                                      description.

   c     output arguments: none

7.5.7. echrtr Subroutine (Getting Information About Element Characteristics)

*deck
subroutine echrtr (iott,elcdn,ielc,kerr)
c   primary function: collect all element characteristics based on
   ityp,jtyp, and keyopts

*** Notice - This file contains ANSYS Confidential information ***

typ=int,dp,log,chr,dcp   siz=sc,ar(n),func   intent=in,out,inout

input arguments:
  variable (typ,siz,intent)      description
  iott (int,sc,in)              - printout file
  ielc (int,ar(IELCSZ),inout)   - input element characteristics
  in positions 1 to 20.
  (itype, jstif, keyopts, etc.)

output arguments:
  elcdn (chr,sc,out)           - element descriptive name as character
  string
  ielc (int,ar(IELCSZ),inout)   - input element characteristics
  in positions 21 to 150.
  (kdim, ishap, idegen, etc.)
  see elccmt for a full list
  kerr (int,sc,out)            - error flag
  = 0 - no errors
  = 1 - errors

7.5.8. etysel Subroutine (Selecting, Unselecting, Deleting, or Inverting an Element Type)

*deck,etysel
   subroutine etysel (itypi,ksel)
c *** primary function: to select, unselect, delete, or invert an
element type.
c *** secondary functions: none.
c
*** Notice - This file contains ANSYS Confidential information ***

typ=int,dp,log,chr,dcp   siz=sc,ar(n),func   intent=in,out,inout

input arguments:
  variable (typ,siz,intent)      description
  itypi (int,sc,in)             - element type number
  = 0 - all element types
  ksel (int,sc,in)              - type of operation to be performed.
  = 0 - delete element type.
  = 1 - select element type.
  =-1 - unselect element type.
  = 2 - invert element type.

output arguments:
none.

7.5.9. mpinqr Function (Getting Information About a Material Property)

*deck,mpinqr
   function mpinqr (mat,iprop,key)
c *** primary function: get information about a material property.
c
*** Notice - This file contains ANSYS Confidential information ***

input arguments:
  mat (int,sc,in)              - material number
  should be 0 for key=11,
  DB_NUMDEFINED(12),
  DB_MAXDEFINED(14), and
  DB_MAXRECLENG(15)
7.5.10. mpget Function (Getting a Material Property Table)

*deck, mpget
   function mpget (mat, iprop, temp, prop)
   c *** primary function:    get a material property table.
   c *** Notice - This file contains ANSYS Confidential information ***
   c typ=int, dp, log, chr, dcp siz=sc, ar(n), func intent=in, out, inout
   c input arguments:
   c variable (typ, siz, intent) description
   c mat      (int, sc, in)       - material number
   c iprop    (int, sc, in)       - property reference number:
   c ---- MP command labels --------
   c EX  = 1, EY  = 2, EZ  = 3, NUXY= 4, NUYZ= 5, NUXZ= 6, GXY = 7, GYZ = 8
   c GXZ = 9, ALPX=10, ALPY=11, ALPZ=12, DENS=13, MU =14, DAMP=15, KXX =16
   c KYY =17, KZZ =18, RSVX=19, RSVY=20, RSVZ=21, C =22, HF =23, VISC=24
   c EMIS=25, ENTH=26, LSST=27, PRXY=28, PRYZ=29, PRXZ=30, MURX=31, MURY=32
   c MURZ=33, PERX=34, PERY=35, PERZ=36, MGXX=37, MGYX=38, MGZX=39, EGXX=40
   c EGYY=41, EGZZ=42, SBXX=43, SBKY=44, SBKZ=45, SONC=46, SLIM=47, ELIM=48
   c USR1=49, USR2=50, USR3=51, USR4=51, FLUI=53, ORTH=54, CABL=55, RIGI=56
   c HGLS=57, BM =58, QRAT=59, REFT=60, CTEX=61, CTEY=62, CTEZ=63, THSX=64,
   c THSY=65, THSZ=66, DMPR=67, LSSM=68, -69, -70, -71, -72,
   c -73, -74, -75, -76, -77, -78, -79, -80
   c (see mpinit for uncommented code and for TB command information)
   c key      (int, sc, in)       - key as to the information needed
   c about material property.
   c = DB_SELECTED(1) - return select status:
   c - mpinqr = 0 - material prop is undefined.
   c - 1 - material prop is selected.
   c = DB_NUMDEFINED(12) - number of defined material properties
   c = DB_MAXDEFINED(14) - highest material property number defined
   c = DB_MAXRECLENG(15) - maximum record length (dp words)
   c = 2 - return length (dp words)
   c = 3 - return number of temp. values
   c = 11 - return void percent (integer)
   c output arguments:
   c mpinqr  (int, func, out)    - returned value of mpinqr is based on
   c setting of key.

7.5.10. mpget Function (Getting a Material Property Table)
7.5.11. **mpput Subroutine (Storing a Material Property Table)**

*deck,mpput
subroutine mpput (mat,iprop,ntab,temp,prop)
c *** primary function: store material property tables.
c *** Notice - This file contains ANSYS Confidential information ***
c input arguments:
c mat (int,sc,in) - material number.
c iprop (int,sc,in) - property reference number:
c ---- MP command labels -------
c EX = 1, EY = 2, EZ = 3, NUXY= 4, NUYZ= 5, NUXZ= 6, GXY = 7, GYZ = 8
c GXZ = 9, ALPX=10, ALFY=11, ALPZ=12, DENS=13, MU -14, DAMP=15, KXX -16
c KYY =17, KZZ =18, RSVX=19, RSVY=20, RSVZ=21, C -22, HF -23, VISC=24
c EMIS=25, ENTH=26, LSST=27, PRXY=28, PRYZ=29, PRXZ=30, MURX=31, MURY=32
c MURZ=33, PERX=34, PERY=35, PERZ=36, MGXX=37, MGYY=38, MGZZ=39, EGXX=40
c EGYY=41, EGZZ=42, SBXX=43, SBKY=44, SBKZ=45, SONC=46, SLIM=47, ELIM=48
c USRl=49, USR2=50, USR3=51, USR4=51, FLUI=53, ORTH=54, CABL=55, RIGI=56
c HGLS=57, BM =58, QRAT=59, REFT=60, CTEX=61, CTEY=62, CTEZ=63, THSX=64,
c THSY=65, THSZ=66, DMPR=67, LSSM=68, -69, -70, -71, -72, -73, -74, -75, -76, -77,
-78, -79, -80
ntab (int,sc,in) - number of entries in the table (1 to 100)
tem (dp,ar(ntab),in) - temperature vector (ascending)
prp (dp,ar(ntab),in) - property vector
c output arguments:
c none.

7.5.12. **mpdel Subroutine (Deleting a Material Property Table)**

*deck,mpdel
subroutine mpdel (mat,iprop)
c *** primary function: delete material property tables.
c *** Notice - This file contains ANSYS Confidential information ***
c input arguments:
c mat (int,sc,in) - material number.
c iprop (int,sc,in) - property reference number:
c ---- MP command labels -------
c EX = 1, EY = 2, EZ = 3, NUXY= 4, NUYZ= 5, NUXZ= 6, GXY = 7, GYZ = 8
c GXZ = 9, ALPX=10, ALFY=11, ALPZ=12, DENS=13, MU -14, DAMP=15, KXX -16
c KYY =17, KZZ =18, RSVX=19, RSVY=20, RSVZ=21, C -22, HF -23, VISC=24
c EMIS=25, ENTH=26, LSST=27, PRXY=28, PRYZ=29, PRXZ=30, MURX=31, MURY=32
c MURZ=33, PERX=34, PERY=35, PERZ=36, MGXX=37, MGYY=38, MGZZ=39, EGXX=40
c EGYY=41, EGZZ=42, SBXX=43, SBKY=44, SBKZ=45, SONC=46, SLIM=47, ELIM=48
c USRl=49, USR2=50, USR3=51, USR4=51, FLUI=53, ORTH=54, CABL=55, RIGI=56
ntab (int,sc,in) - number of entries in the table (1 to 100)
tem (dp,ar(ntab),in) - temperature vector (ascending)
prp (dp,ar(ntab),in) - property vector
c output arguments: none.

7.5.13. **rlinqr Function (Getting Information About a Real Constant Set)**

*deck,rlinqr
function rlinqr (nreal,key)
c *** primary function: get information about a real constant set
7.5.14. rlget Function (Getting Real Constant Data)

*deck,rlget

function rlget (nreal,rtable)

7.5.15. rlsel Subroutine (Selecting or Deleting a Real Constant Set)

*deck,rlsel

subroutine rlsel (nreal,ksel)
7.5.16. csyiqr Function (Getting Information About a Coordinate System)

*deck,csyiqr
  function csyiqr (ncsy,key)
  c *** primary function: get information about a coordinate system
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c   ncsy   (int,sc,in) - coordinate system reference number
  c       should be zero for key= DB_NUMDEFINED
  c       or DB_MAXDEFINED
  c   key    (int,sc,in) - information flag.
  c       = DB_SELECTED - return status:
  c                       csyiqr = 0 - coordinate system is not defined
  c                      -1 - coordinate system is not selected
  c                      1 - coordinate system is selected
  c       = DB_NUMDEFINED - number of defined coordinate systems
  c       = DB_MAXDEFINED - maximum coordinate system reference
  c output arguments:
  c   csyiqr   (int,func,out) - the returned value of csyiqr is based on
  c       setting of key.

7.5.17. csyget Function (Getting a Coordinate System)

*deck,csyget
  function csyget (ncsy,csydpx,csyinx)
  c *** primary function: get a coordinate system
  c *** secondary functions: none
  c *** Notice - This file contains ANSYS Confidential information ***
  c NOTE: As a time-saving device, this routine will not fetch the coordinate
  c system data from the database (an expensive operation)
  c if ncsy = csyinx(4), as this would indicate that the data is current.
  c If you wish to force this routine to fetch coordinate system data (in
  c the case of loading a local array, for example), you MUST set
  c ncsy != csyinx(4) before function call.
  c   typ=int,dp,log,chr,dcp   siz=sc,ar(n),func    intent=in,out,inout
  c input arguments:
  c   variable (typ,siz,intent)    description                  csycom name
  c   ncsy     (int,sc,in)       - coordinate system number
  c   csyinx(4) (int,sc,inout)   - coordinate system number     csyact
  c output arguments:
  c   csydpx   (dp,ar(18),out) - transformation matrix
  c       (1-9) - origin (XC, YC, ZC)
  c       (10-12) - coordinate system parameters cparm
  c       (13-14) - coordinate system parameters cparm2
  c       (15)    - spare
  c       (16-18) - defining angles
  c   csyinx   (int,ar(6),out) - theta, phi singularity keys
  c       (1-2) - coordinate system type icdysys
  c       (3) - coordinate system number csyact
  c       (4) - coordinate system number csyact
  c       (5) - spare
  c       (6) - spare
  c   csyget   (int,func,out) - status of coordinate system
  c       = 0 - coordinate system exists
  c       = 1 - coordinate system doesn't exist
7.5.18. csyput Subroutine (Storing a Coordinate System)

```c
*deck,cspyut
  subroutine csyput (ncsy,csydpx,csyinx)
  c *** primary function: store a coordinate system
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c   ncsy    (int,sc,in)       - coordinate system number
  c   csydpx  (dp,ar(18),out)
  c                          csydpx(1-9) - transformation matrix
  c                                (10-12) - origin (XC, YC, ZC)
  c                                (13-14) - coordinate system parameters cparm
  c                                cparm2
  c                                (15)  - spare
  c                                (16-18) - defining angles
  c   csyinx (int,ar(6),out)
  c                          csyinx(1-2) - theta, phi singularity keys
  c                                (3)   - coordinate system type icdys
  c                                (4)   - coordinate system number csyact
  c                                (5)   - spare
  c                                (6)   - spare
  c output arguments: none
```

7.5.19. csydel Subroutine (Deleting a Coordinate System)

```c
*deck,cspydel
  subroutine csydel (ncsy)
  c *** primary function: delete a coordinate system
  c *** secondary functions: none
  c c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c   variable (typ,siz,intent)  description
  c   ncsy    (int,sc,in)       - coordinate system number
  c output arguments:
  c none
```

7.5.20. userac Subroutine (Demonstrates Use of Element Attribute Routines)

See Section 6.1.2.8: Subroutine userac (Accessing Element Information) for an example that demonstrates how to use the userac subroutine to extract information about an element type and element real constants from the ANSYS database. You can find this subroutine on your ANSYS distribution media.

7.6. Coupling and Constraint Routines

7.6.1. cpinqr Function (Getting Information About a Coupled Set)

```c
*deck,cpinqr
  function cpinqr (nce,key)
  c *** primary function: get information about a coupled set
  c *** secondary functions: none
  c c *** Notice - This file contains ANSYS Confidential information ***
  c c typ=int,dp,log,chr,dcp  siz=sc,ar(n),func  intent=in,out,inout
```
Chapter 7: Accessing the ANSYS Database

7.6.2. cpget Function (Getting a Coupled Set)

*deck,cpget
  function cpget (ncp,ieqn)
  c *** primary function:    get a coupled set
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c     ncp      (int,sc,in)                  - coupled set number
  c output arguments:
  c     cpget         (int,func,out)        - number of nodes in list
  c     ieqn          (int,ar(cpget+2),out) - coupled set info:
  c                       ieqn(1:cpget) - list of coupled nodes
  c                       ieqn(cpget+1) - set degree of freedom
  c                       ieqn(cpget+2) - number of nodes in list
                          (copy of return value)

7.6.3. cpput Subroutine (Storing a Coupled Set)

*deck,cpput
  subroutine cpput (ncp,n,ieqn)
  c *** primary function:    store a coupling set
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c     ncp       (int,sc,in)        - coupled set number
  c     n         (int,sc,in)        - number of nodes in coupled set
  c     ieqn      (int,ar(n+2),in)   - info for storage
  c                       ieqn(1:n) - list of coupled nodes
  c                       ieqn(n+1) - degree of freedom label for set
  c (ieqn(n+2) is inout) ieqn(n+2) - number of nodes in coupled set
                          (copy of n)
  c output arguments:
7.6.4. cpsel Subroutine (Selecting or Deleting a Coupled Set)

*deck,cpsel
  subroutine cpsel (ncpi,ksel)
  c *** primary function:    select or delete a coupled set
  c *** secondary functions: none
  c
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  c     typ=int,dp,log,chr,dcp   siz=sc,ar(n),func    intent=in,out,inout
  c
  c input arguments:
  c     variable (typ,siz,intent)    description
  c     ncpi      (int,sc,in)       - coupled set number
  c     ksel      (int,sc,in)       - select/delete flag
  c                                   = 0 - delete coupled set
  c                                   = 1 - select coupled set
  c
  c output arguments:
  c     none
  c

7.6.5. ceinqr Function (Getting Information About a Constraint Equation Set)

*deck,ceinqr
  function ceinqr (nce,key)
  c *** primary function:    get information about a constraint equation set
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  c input arguments:
  c     nce         (int,sc,in)       - constraint equation number
  c     key         (int,sc,in)       - inquiry key:
  c                                     should be zero for key=11, DB_NUMDEFINED,
  c                                      DB_NUMSELECTED, DB_MAXDEFINED, and
  c                                      DB_MAXRECLENG
  c     = DB_SELECTED - return select status
  c     ceinqr = 1 - equation is selected
  c     = 0 - equation is undefined
  c     =-1 - equation is unselected
  c     = DB_NUMDEFINED - return number of defined constraint equations
  c     = DB_NUMSELECTED - return number of selected constraint equations
  c     = DB_MAXDEFINED - return number of highest numbered constraint
  c        equation defined
  c     = DB_MAXRECLENG - return length of longest constraint equation set
  c        (max record length)
  c     = 2 - return length (data units)
  c     = 3 - return layer number
  c     = 4 - address of first data word
  c     = 11 - return void percent (integer)
  c     = 16 - return location of next record
  c     = -1 - return master dof for this eqn
  c
  c output arguments:
  c     ceinqr   (int,func,out)    - the returned value of ceinqr is based on
  c                                       setting of key
  c

7.6.6. ceget Function (Getting an Constraint Equation)

*deck,ceget
  function ceget (nce,ieqn,deqn)
  c *** primary function:    get a constraint equation
  c *** Notice - This file contains ANSYS Confidential information ***
7.6.7. ceput Subroutine (Storing a Constraint Equation)

*deck,ceput
  subroutine ceput (nce,n,ieqn,deqn)
  c *** primary function: store a constraint equation
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c   nce       (int,sc,in)       - constraint equation set number
  c   n         (int,sc,in)       - number of degrees of freedom in set
  c   ieqn      (int,ar(n+1),in)  - integer info
  c                     ieqn(1:n) - node*32+dof for each dof in set
  c                     ieqn(n+1) - number of dof in set (copy of n above)
  c   deqn      (dp,ar(n+1),in)   - dp info
  c                     deqn(1:n) - coefficients of each dof in set
  c                     deqn(n+1) - constant term
  c
  c output arguments: none

7.6.8. cesel Subroutine (Deleting or Selecting a Constraint Equation)

*deck,cesel
  subroutine cesel (ncei,ksel)
  c *** primary function: select or delete a constraint equation
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c   ncei      (int,sc,in)       - constraint equation number
  c   ksel      (int,sc,in)       - select/delete flag
  c                           = 0 - delete equation
  c                           = 1 - select equation
  c
  c output arguments: none

7.7. Nodal Loading Routines

7.7.1. disiqr Function (Getting a Information About Constraints)

*deck,disiqr
  function disiqr (node,key)
  c *** primary function: get information about constraints
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c   node      (int,sc,in)       - node number for inquire.
  c   key       (int,sc,in)       - key as to the information needed
7.7.2. disget Function (Getting a Constraint from the Database)

*deck,disget
function disget (inode,idf,value)
  
  c *** primary function: get a constraint from the data base (in raw form)

  c *** Notice - This file contains ANSYS Confidential information ***

  c input arguments:
  c variable (typ,siz,intent) description
  c inode (int,sc,in) - node number (negative value for no partabeval)
  c idf (int,sc,in) - reference number for the DOF: (1-32)
  c UX = 1, UY = 2, UZ = 3, ROTX= 4, ROTY= 5, ROTZ= 6, AX = 7, AY = 8
  c AZ = 9, VX =10, VY =11, VZ =12
  c PRES=19, TEMP=20, VOLT=21, MAG =22, ENKE=23, ENDS=24
  c EMF =25, CURR=26 (missing entries are spares)

  c output arguments:
  c disget (int,func,out) - status of constraint.
  c = 0 - no constraint on this node for this DOF
  c = 4 - this node has a constraint defined for this DOF
  c = -4 - this node has a pseudo-support defined for this DOF
  c value (dp,ar(4),out) - constraint values
  c value(1-2) - (real,imag) values of present settings
  c value(3-4) - (real,imag) values of previous settings

7.7.3. disput Subroutine (Storing a Constraint at a Node)

*deck,disput
subroutine disput (node,idf,value)

  c *** primary function: store a constraint at a node.

  c *** Notice - This file contains ANSYS Confidential information ***

  c input arguments:
  c node (int,sc,in) - node number
  c idf (int,sc,in) - reference number of DOF: (1-32)
  c UX = 1, UY = 2, UZ = 3, ROTX= 4, ROTY= 5, ROTZ= 6, AX = 7, AY = 8
  c AZ = 9, VX =10, VY =11, VZ =12
  c PRES=19, TEMP=20, VOLT=21, MAG =22, ENKE=23, ENDS=24
  c EMF =25, CURR=26 (missing entries are spares)

  c value (dp,ar(2),in) - (real,imag) values for constraint

  c output arguments: none.

7.7.4. disdel Subroutine (Deleting a Constraint at a Node)

*deck,disdel
subroutine disdel (node,idf)
7.7.5. foriqr Function (Getting Information About Nodal Loads)

*deck, foriqr
function foriqr (node, key)

7.7.6. forget Function (Getting a Constraint from the Database)

*deck, forget
function forget (inode, idf, value)

7.7.7. forput Subroutine (Storing a Nodal Load at a Node)

*deck, forput
subroutine forput (node, idf, value)
7.7.9. ntpiqr Function (Getting Information About a Nodal Temperature)

*deck, ntpiqr
  function ntpiqr (node, key)
  c *** Notice - This file contains ANSYS Confidential information ***
  c typo-int, dp, log, chr, dcp size-sc, ar(n), func intent=in, out, inout
  c input arguments:
  c variable (typ, siz, intent) description
  c node (int, sc, in) - node number
  c key (int, sc, in) - should be zero for key=2
  c output arguments:
  c ndinqr (int, func, out) - the returned value of ndinqr is based on
  c setting of key.
7.7.10. ntpget Function (Getting a Specified Nodal Temperature)

```c
*deck,ntpget
  function ntpget (node,tmp)
  c *** primary function: get specified nodal heat generation (in raw form)
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  c     input arguments:
  c        node     (int,sc,in)       - node number
  c
  c     output arguments:
  c        ntpget   (int,func,out)    - heat generation status of node.
  c                                     = 0 - nodal heat generation undefined
  c                                     = 1 - nodal heat generation is defined
  c        tmp       (dp,ar(2),out)    - the nodal heat generation (new,old).
```

7.7.11. ntpput Subroutine (Storing a Nodal Temperature)

```c
*deck,ntpput
  subroutine ntpput (node,temp)
  c *** primary function: store nodal temperature.
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  c     input arguments:
  c        node    (int,sc,in)       - node number
  c        temp    (dp ,sc,in)       - nodal temperature
  c
  c     output arguments: none.
```

7.7.12. ntpdel Subroutine (Deleting a Nodal Temperature)

```c
*deck,ntpdel
  subroutine ntpdel (node)
  c *** primary function: delete node temperatures.
  c *** Notice - This file contains ANSYS Confidential information ***
  c
c     typ=int,dp,log,chr,dcp   siz=sc,ar(n),func    intent=in,out,inout
  c
  c     input arguments:
  c        node    (int,sc,in)       - node number
  c
  c     output arguments: none.
```

7.7.13. nhgiqr Function (Getting Information About Nodal Heat Generations)

```c
*deck,nhgiqr
  function nhgiqr (node,key)
  c *** primary function: get information about nodal heat generations
  c *** Notice - This file contains ANSYS Confidential information ***
  c
c     typ=int,dp,log,chr,dcp   siz=sc,ar(n),func    intent=in,out,inout
  c
  c     input arguments:
  c        node     (int,sc,in)       - node number
  c
  c     variable (typ,siz,intent)    description
  c        node     (int,sc,in)       - node number
  c
  c     output arguments:
  c        key      (int,sc,in)       - key for operation
```

Programmer's Manual for ANSYS . ANSYS Release 11.0 . 002328 . © SASIP Inc. All rights reserved.
Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.
7.7.14. nhgget Function (Getting a Nodal Heat Generation)

*deck, nhgget
    function nhgget (node, hg)
    c *** primary function: get specified nodal heat generation (in raw form)
    c *** Notice - This file contains ANSYS Confidential information ***
    c typ=int, dp, log, chr, dcp  siz=sc, ar(n), func  intent=in, out, inout
    c input arguments:
    c variable (typ, siz, intent)  description
    c node (int, sc, in)           - node number
    c output arguments:
    c nhgget (int, func, out)      - heat generation status of node.
    c hg (dp, ar(2), out)          - the nodal heat generation (new, old).

7.7.15. nhgput Subroutine (Storing Nodal Heat Generation)

*deck, nhgput
    subroutine nhgput (node, hg)
    c *** primary function: store nodal heat generation.
    c *** Notice - This file contains ANSYS Confidential information ***
    c input arguments:
    c node (int, sc, in)           - node number
    c hg (dp, sc, in)              - nodal heat generation
    c output arguments: none.

7.7.16. nhgdel Subroutine (Deleting a Nodal Heat Generation)

*deck, nhgdel
    subroutine nhgdel (node)
    c *** primary function: delete nodal heat generations.
    c *** Notice - This file contains ANSYS Confidential information ***
    c typ=int, dp, log, chr, dcp  siz=sc, ar(n), func  intent=in, out, inout
    c input arguments:
    c variable (typ, siz, intent)  description
    c node (int, sc, in)           - node number
    c output arguments: none.

7.7.17. nfuiqr Function (Getting Information About Nodal Fluences)

*deck, nfuiqr
    function nfuiqr (node, key)
c *** primary function: get information about nodal fluences


c *** Notice - This file contains ANSYS Confidential information ***

typ=int,dp,log,chr,dcp siz=sc,ar(n),func intent=in,out,inout

input arguments:

variable (typ,siz,intent) description
node (int,sc,in) - node number
key (int,sc,in) - key for operation

output arguments:
nfuiqr (int,func,out) - the returned value of nfuiqr is based on
                          setting of key.

7.7.18. nfuget Function (Getting a Nodal Fluence)

*deck,nfuget
  function nfuget (node,fluen)
  c *** primary function: get specified nodal fluence.
  c *** Notice - This file contains ANSYS Confidential information ***

input arguments:
node (int,sc,in) - node number

output arguments:
nfuiqr (int,func,out) - fluence status of node.
fluen (dp,ar(2),out) - the nodal fluences (new,old).

7.7.19. nfuput Subroutine (Storing a Nodal Fluence)

*deck,nfuput
  subroutine nfuput (node,fluen)
  c *** primary function: store nodal fluence.
  c *** Notice - This file contains ANSYS Confidential information ***

input arguments:
node (int,sc,in) - node number
fluen (dp,sc,in) - nodal fluence

output arguments: none.

7.7.20. nfudel Subroutine (Deleting a Nodal Fluence)

*deck,nfudel
  subroutine nfudel (node)
  c *** primary function: delete node fluences.
  c *** Notice - This file contains ANSYS Confidential information ***

typ=int,dp,log,chr,dcp siz=sc,ar(n),func intent=in,out,inout

input arguments:

variable (typ,siz,intent) description
node (int,sc,in) - node number

output arguments:
7.7.21. ndciqr Function (Getting Information About Nodal Current Densities)

*deck,ndciqr
  function ndciqr (node,key)
  c *** primary function: get information about nodal current densities
  c *** Notice - This file contains ANSYS Confidential information ***
  c     typ=int,dp,log,chr,dcp siz=sc,ar(n),func intent=in,out,inout
  c     input arguments:
  c       variable (typ,siz,intent)     description
  c          node     (int,sc,in)       - node number
  c             should be zero for key=2
  c          key      (int,sc,in)       - key for operation
  c             = 1 - return nodal current status:
  c                ndciqr = 0 - no current density defined for this node
  c                = 1 - node has a current density defined
  c             = 2 - total number of nodal current densities defined on model
  c     output arguments:
  c       ndciqr   (int,func,out)    - the returned value of ndciqr is based on setting of key.

7.7.22. ndcget Function (Getting a Nodal Current Density)

*deck,ndcget
  function ndcget (node,currd)
  c *** primary function: get specified nodal current density.
  c *** Notice - This file contains ANSYS Confidential information ***
  c     input arguments:
  c       node     (int,sc,in)       - node number
  c     output arguments:
  c       ndcget   (int,func,out)    - current density status of node.
  c             = 0 - node has no current density defined
  c             = 1 - node has a current density defined
  c       currd    (dp,ar(4,2),out)  - the node current density (new,old).

7.7.23. ndcput Subroutine (Storing a Nodal Current Density)

*deck,ndcput
  subroutine ndcput (node,currd)
  c *** primary function: store nodal current density.
  c *** Notice - This file contains ANSYS Confidential information ***
  c     input arguments:
  c       node    (int,sc,in)       - node number
  c       currd   (dp,ar(4),in)    - nodal current densities
  c     output arguments: none.

7.7.24. ndcdel Subroutine (Deleting a Nodal Current Density)

*deck,ndcdel
  subroutine ndcdel (node)
  c *** primary function: delete nodal current densities
Chapter 7: Accessing the ANSYS Database

7.7.25. nvdiqr Function (Getting Information About Nodal Magnetic Virtual Displacements)

*deck,nvdiqr
function nvdiqr (node,key)
c *** primary function: get information about nodal mag virtual displs

7.7.26. nvdget Function (Getting a Nodal Magnetic Virtual Displacement)

*deck,nvdget
function nvdget (node,virtd)
c *** primary function: get specified nodal magnetic virtual displacement

7.7.27. nvdput Subroutine (Storing a Nodal Virtual Displacement)

*deck,nvdput
subroutine nvdput (node,virtd)
c *** primary function: store nodal virtual displacement

Programmer's Manual for ANSYS. ANSYS Release 11.0. 002328. © SASIP, Inc. All rights reserved.
Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.
7.7.28. nvddel Subroutine (Deleting a Nodal Virtual Displacement)

*deck, nvddel
    subroutine nvddel (node)
    c *** primary function: delete nodal virtual displacements.
    c *** Notice - This file contains ANSYS Confidential information ***
    c
    c     typ=int,dp,log,chr,dcp siz=sc,ar(n),func intent=in,out,inout
    c
    c     input arguments:
    c     variable (typ,siz,intent)    description
    c        node    (int,sc,in)       - node number
    c
    c     output arguments:
    c        none.

7.8. Element Loading Routines

7.8.1. epriqr Function (Getting Information About Element Pressure/Convection)

*deck, epriqr
    function epriqr (ielem,iface,key)
    c *** primary function: get information about element pressure/convection
    c *** Notice - This file contains ANSYS Confidential information ***
    c
    c     input arguments:
    c        ielem    (int,sc,in)       - element number
        iface    (int,sc,in)       - face number for inquire (0-6)
        key      (int,sc,in)       - key as to the information needed
            = 1              - return pressure mask for element
            = 5              - return number of pressures for this
                                element face
            = DB_NUMDEFINED,  - return value is based on setting of iface
            = DB_MAXDEFINED  - return the maximum number of element
                                pressures on any element (max record
                                length)
            iface = 0 - return number of surface loads defined
            = 1-6 - return number of pressure loads defined for this
                                element.
        NOTE: only 1-6 is valid, but this
        routine simply checks that iface is in
        the range. The actual value of iface
        does not matter in this case.
            = DB_MAXRECLENG
    c
    c     output arguments:
    c        epriqr   (int,func,out)    - the returned value of epriqr is based on
                                        setting of key.

7.8.2. eprget Function (Getting an Element Face Pressure)

*deck, eprget
    function eprget (elem,iface,value)
    c *** primary function: get an element face pressure
    c *** Notice - This file contains ANSYS Confidential information ***
Chapter 7: Accessing the ANSYS Database

7.8.3. eprput Subroutine (Storing an Element Face Pressure)

*deck,eprput
    subroutine eprput (ielem,iface,nval,value)
    c *** primary function: store an element face pressure.
    c *** Notice - This file contains ANSYS Confidential information ***
    c input arguments:
    c   ielem (int,sc,in) - element number for operation.
    c   iface (int,sc,in) - face number (1-6)
    c   nval (int,sc,in) - number of values to put
    c   value (dp ,ar(nval),in) - the element pressures (real,imag) at each face
    c output arguments: none.

7.8.4. epdel Subroutine (Deleting an Element Pressure/Convection)

*deck,epdel
    subroutine eprdel (ielem,iface)
    c *** primary function: delete a pressure/convection on an element
    c *** Notice - This file contains ANSYS Confidential information ***
    c input arguments:
    c   ielem (int,sc,in) - element number
    c   iface (int,sc,in) - face number
    c   = 0 - delete all pressures on this element
    c   = 1-6 - delete pressure on this face
    c output arguments: none.

7.8.5. ecviqr Function (Getting Information About Element Convections)

*deck,ecviqr
    function ecviqr (ielem,iface,key)
    c *** primary function: get information about element convections
    c *** Notice - This file contains ANSYS Confidential information ***
    c input arguments:
    c   ielem (int,sc,in) - element number for inquire
    c   iface (int,sc,in) - face number
    c   key (int,sc,in) - key as to the information needed
    c   = 1 - return convection mask for element
    c   = 5 - return number of convections for this element face
    c output arguments: none.
7.8.6. ecvget Function (Getting an Element Face Convection)

*deck,ecvget

function ecvget (elem,iface,value)

*** primary function: get an element face convection (in raw form)

*** Notice - This file contains ANSYS Confidential information ***

input arguments:

c         elem     (int,sc,in)       - element number

c         iface    (int,sc,in)       - face number (1-6)

output arguments:

c         ecvget   (int,func,out)    - status of element.

c         value    (dp ,ar(*),out)   - the element convections

NOTE: Two values at each node of an element face: if loading is a convection, the first first value is the film coefficient and the second value is the bulk temperature. If loading is a heat flux, the first value is the heat flux, and the second value is a large number (2**100)

7.8.7. ecvput Subroutine (Storing an Element Face Convection)

*deck,ecvput

subroutine ecvput (ielem,iface,nval,value)

*** primary function: store an element face convection.

*** Notice - This file contains ANSYS Confidential information ***

input arguments:

variable (typ,siz,intent)     description

c         ielem    (int,sc,in)       - element number

c         iface    (int,sc,in)       - face number (1-6)

c         nval     (int,sc,in)       - number of values to put

c         value    (dp ,ar(nval),in) - the element convections.

NOTE: Two values at each node of an element face: if loading is a convection, the first first value is the film coefficient and the second value is the
7.8.8. ecvdel Subroutine (Deleting a Convection on an Element)

*deck,ecvdel
subroutine ecvdel (ielem,iface)
c *** primary function: delete a convection on an element
c *** Notice - This file contains ANSYS Confidential information***
c     typ=int,dp,log,chr,dcp siz=sc,ar(n),func intent=in,out,inout
c     input arguments:
c     variable (typ,siz,intent) description
     ielem (int,sc,in) - element number.
     iface (int,sc,in) - face number
     = 0 - delete all convections on this element
     = 1-6 - delete convections on this face
     output arguments:
c     none.

7.8.9. etpiqr Function (Getting Information About Element Temperatures)

*deck,etpiqr
function etpiqr (ielem,key)
c *** primary function: get information about element temperatures.
c *** Notice - This file contains ANSYS Confidential information***
c     input arguments:
c     ielem (int,sc,in) - element number
     Should be 0 for key=11, DB_NUMDEFINED, DB_MAXDEFINED, and DB_MAXRECLENG
     key (int,sc,in) - information flag.
     = DB_SELECTED - return status:
        etpiqr = 0 - element has no temperatures
        = 1 - element has temperatures defined
     = DB_NUMDEFINED - return number of temperatures defined for this element (rec length)
     = DB_MAXDEFINED - return number of temperatures defined in model
     = DB_MAXRECLENG - return maximum number of temperatures defined for any element (max rec length)
     = 2 - return length (dp words)
     = 3 - return layer number (for cross reference files return number of entities)
     = 4 - return address of first data word
     = 5 - return length (dp words)
     = 6 - return compressed record number.
     = 11 - return void percent (integer)
     = 16 - return location of next record (this increments the next record count)
     = 18 - return type of file.
        etpiqr = 0 - integer
        = 1 - double precision
        = 2 - real
        = 3 - complex
        = 4 - character*8
        = 7 - index
     = 19 - return virtual type of file.
        etpiqr = 0 - fixed length (4.4 form)
        = 1 - indexed variable length
7.8.10. etpget Function (Getting an Element Temperature)

*deck,etpget
  function etpget (ielem,tem)
  c *** primary function: get element temperatures (in raw form)
  c *** Notice - This file contains ANSYS Confidential information ***
  c  input arguments:
  c    ielem    (int,sc,in)       - element number
  c  output arguments:  etpget   (int,func,out)    - status of element.
  c    etpget   (int,func,out)    - 0 - this element has no element
  c    > 0 - number of element temperatures
  c    tem     (dp,ar(n,2),out)  - the element temperatures (new,old).
  c  NOTE THAT TEM MUST DOUBLE THE NUMBER OF DESIRED
  c  TEMPERATURES IN THE CALLING ROUTINE!
  c  NOTE: If a value is not defined (i.e.,
  c  defaults to TUNIF), value will be a
  c  very small number (2**-100)

7.8.11. etpput Subroutine (Storing an Element Temperature)

*deck,etpput
  subroutine etpput (ielem,n,temp)
  c *** primary function: store element temperatures.
  c *** Notice - This file contains ANSYS Confidential information ***
  c  input arguments:
  c    ielem    (int,sc,in)       - element number
  c    n        (int,sc,in)       - number of element temperature values
  c    temp     (dp ,ar(n),in)    - element temperatures.
  c  output arguments:  none.
  c  NOTE: If a value is not defined (i.e.,
  c  defaults to TUNIF), a very small
  c  number should be used (2***-100)

7.8.12. etpdel Subroutine (Deleting an Element Temperature)

*deck,etpdel
  subroutine etpdel (ielem)
  c *** primary function: delete element temperatures.
  c *** Notice - This file contains ANSYS Confidential information ***
  c  input arguments:
  c    ielem    (int,sc,in)       - element number
  c  output arguments:  none.
### 7.8.13. ehgiqr Function (Getting Information About Element Heat Generation)

```fortran
*deck,ehgiqr
  function ehgiqr (ielem,key)
  c *** primary function: get information about element heat generations.
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  input arguments:
  c    ielem  (int,sc,in)       - element number
  c                                      should be 0 for key=11, DB_NUMDEFINED,
  c                                      DB_MAXDEFINED, and DB_MAXRECLENG
  c    key    (int,sc,in)       - information flag.
  c           DB_SELECTED - return status:
  c                      ehgiqr = 0 - heat generation is undefined
  c                      = 1 - heat generation is defined
  c           DB_NUMDEFINED  - return number of defined heat generations
  c                                      in model
  c           DB_MAXRECLENG  - return maximum number of heat generations
  c                                       on any element (max rec length)
  c           2 - return length (dp words)
  c           3 - return layer number (for cross reference files return
  c                                      number of entities)
  c           4 - return address of first data word
  c           5 - return length (record type units)
  c           6 - return compressed record number.
  c           11 - return void percent (integer)
  c           16 - return location of next record (this increments the
  c                                      next record count)
  c           18 - return type of file.
  c                      ehgiqr = 0 - integer
  c                      = 1 - double precision
  c                      = 2 - real
  c                      = 3 - complex
  c                      = 4 - character*8
  c                      = 7 - index
  c           19 - return virtual type of file.
  c                      ehgiqr = 0 - fixed length (4.4 form)
  c                      = 1 - indexed variable length
  c                      (layer data)
  c                      = 2 - xref data tables
  c                      = 3 - bitmap data (for 32 data
  c                                      item packed records)
  c                      = 4 - data tables (three
c                                      dimensional arrays)
  c
  output arguments:
  c    ehgiqr  (int,func,out)    - the returned value of ehgiqr is based on
  c                                      setting of key.
```

### 7.8.14. ehgget Function (Getting an Element Heat Generation)

```fortran
*deck,ehgget
  function ehgget (ielem,qgen)
  c *** primary function: get element heat generations (in raw form)
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  input arguments:
  c    ielem  (int,sc,in)       - element number
  c
  output arguments:
  c    ehgget  (int,func,out)    - status of element.
  c                      = 0 - heat generations undefined for this
  c                                      element
  c                      > 0 - number of heat generations defined
  c    qgen    (dp ,ar(*),out)   - the element heat generations.
```
7.8.15. ehgput Subroutine (Storing an Element Heat Generation)

```fortran
*deck,ehgput
subroutine ehgput (ielem,n,qgen)
c *** primary function: store element heat generations

    c *** Notice - This file contains ANSYS Confidential information ***

    c input arguments:
    c   ielem      (int,sc,in)       - element number
    c   n          (int,sc,in)       - number of element heat generation values
    c   qgen       (dp ,ar(n),in)    - element heat generations

    c output arguments: none

    c NOTE: If a value is not defined, it will be a very small number (2**-100)
```

7.8.16. ehgdel Subroutine (Deleting an Element Heat Generation)

```fortran
*deck,ehgdel
subroutine ehgdel (ielem)
c *** primary function: delete element heat generations.

    c *** Notice - This file contains ANSYS Confidential information ***

    c input arguments:
    c   ielem (typ,dp,log,chr,dcp) siz=ar(n),func intent=in,out,inout

    c output arguments:
    c   none
```

7.8.17. efuiqr Function (Getting Information About Element Fluences)

```fortran
*deck,efuiqr
function efuiqr (ielem,key)
c *** primary function: get information about element fluences

    c *** Notice - This file contains ANSYS Confidential information ***

    c input arguments:
    c   ielem      (int,sc,in)       - element number or zero (see below)
    c   key        (int,sc,in)       - key as to the information needed
    c                 = 1 or DB_MAXRECLENG - return element fluences info
    c                 for ielem > 0 - return number of fluences for this
    c                 element (record length)
    c                 = 0 - return maximum number of fluences
    c                 defined for any element
    c                 (max rec length)
    c                 = DB_NUMDEFINED,
    c                 = DB_MAXDEFINED - return number of defined fluences
    c                 in model
    c                 NOTE: both DB_NUMDEFINED and DB_MAXDEFINED
    c                 produce the same functionality

    c output arguments:
    c   efuiqr (int,func,out)    - the returned value of efuiqr is based on
    c                 setting of key
```

NOTE: If a value is not defined, it will be a very small number (2**-100)
7.8.18. efuget Function (Getting an Element Fluence)

*deck,efuget
  function efuget (ielem,value)
c *** primary function:    get element fluences.
c *** Notice - This file contains ANSYS Confidential information ***
c input arguments:
c  ielem   (int,sc,in)       - element number

c output arguments:
c  efuget   (int,func,out)    - status of element.
c  value    (dp,ar(*),out)    - element fluences.
  NOTE: If a value is not defined, it will be a very small number (2**-100)

7.8.19. efuput Subroutine (Storing an Element Fluence)

*deck,efuput
  subroutine efuput (ielem,n,value)
c *** primary function:    store element fluences
  NOTE: If a value is not defined, a very small number should be used (2**-100)

7.8.20. efudel Subroutine (Deleting an Element Fluence)

*deck,efudel
  subroutine efudel (ielem)
c *** primary function:    delete element fluences
  NOTE: If a value is not defined, a very small number should be used (2**-100)

7.8.21. edciqr Function (Getting Information About Element Current Densities)

*deck,edciqr
  function edciqr (ielem,key)
c *** primary function:    get information about element current densities
  NOTE: If a value is not defined, a very small number should be used (2**-100)
7.8.22. edcget Function (Getting Element Current Densities)

*deck,edcget
function edcget (ielem,value)
c *** primary function: get element current densities
c *** Notice - This file contains ANSYS Confidential information ***
typ=int,dp,log,chr,dcp siz=sc,ar(n),func intent=in,out,inout

    ielem (int,sc,in) - element number

output arguments:
   edcqr (int,func,out) - the returned value of edcqr is based on setting of key

7.8.23. edcput Subroutine (Storing an Element Current Density)

*deck,edcput
subroutine edcput (ielem,n,value)
c *** primary function: store element current densities
c *** Notice - This file contains ANSYS Confidential information ***

    ielem (int,sc,in)
    n (int,sc,in)
    value (dp,ar(n),in)

output arguments:
   none

NOTE: If a value is not defined, a very small number should be used (2**-100)

7.8.24. edcdel Subroutine (Deleting an Element Current Density)

*deck,edcdel
subroutine edcdel (ielem)
c *** primary function: delete element current densities
c *** Notice - This file contains ANSYS Confidential information ***

    ielem (int,sc,in)

output arguments:  none

NOTE: If a value is not defined, a very small number should be used (2**-100)
7.8.25. evdiqr Function (Getting Information About Element Virtual Displacements)

```fortran
*deck, evdiqr
  function evdiqr (ielem, key)
  c *** primary function: get information about element virt disps
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c   ielem  (int, sc, in)    - element number or zero (see below)
  c   key    (int, sc, in)    - key as to the information needed
  c     = 1 or DB_MAXRECLENG - return element virt disps info
  c        for ielem > 0 - number of virt disps defined for this
  c             element (rec length)
  c     = 0 - maximum number of virt disps defined
  c             for any element (max rec length)
  c     = DB_NUMDEFINED,
  c     = DB_MAXDEFINED - return total number of virt disps defined
  c             in model
  c output arguments:  evdiqr    (int, func, out)    - the returned value of evdiqr is based on
  c                        setting of key
```

7.8.26. evdget Function (Getting an Element Virtual Displacement)

```fortran
*deck, evdget
  function evdget (ielem, value)
  c *** primary function: get element virtual displacements
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c   ielem    (int, sc, in)    - element number
  c output arguments:  evdget     (int, func, out)    - status of element.
  c        = 0 - no virt disps defined for this
  c             element
  c        > 0 - number of element virtual
  c             displacements
  c   value    (dp, ar(*), out)    - element virtual displacements
  c NOTE: If a value is not defined, it will
  c       be a very small number (2**-100)
```

7.8.27. evdput Subroutine (Storing an Element Virtual Displacement)

```fortran
*deck, evdput
  subroutine evdput (ielem, n, value)
  c *** primary function: store element virtual displacements
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c   ielem    (int, sc, in)    - element number
  c   n        (int, sc, in)    - the total number of values
  c   value    (dp, ar(n), in)    - element virtual displacements
  c output arguments:  none
  c NOTE: If a value is not defined, a very
  c       small number should be used (2**-100)
```
7.8.28. eimiqr Function (Getting Information About Element Impedances)

*deck,eimiqr
  function eimiqr (ielem,iface,key)
  c *** primary function: get information about element impedences
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c ielem (int,sc,in) - element number for inquire.
  c should be zero for key=DB_NUMDEFINED, DB_MAXDEFINED or DB_MAXRECLENG
  c iface (int,sc,in) - face number for inquire (0-6)
  c face number is needed for key=5. for other values of key, iface has different meaning (see below)
  c key (int,sc,in) - key as to the information needed
  c = 1 - return impedance mask for element
  c = 5 - return number of impedences for this element face
  c = DB_NUMDEFINED, = DB_MAXDEFINED - return value is based on setting of iface
  c NOTE: both DB_NUMDEFINED and DB_MAXDEFINED produce the same functionality
  c iface = 0 - return number of surface loads defined in model
  c = 1-6 - return number of pressure loads defined for this element. (rec length)
  c NOTE: only 1-6 is valid, but this routine simply checks that iface is in the range. The actual value of iface does not matter in this case.
  c = DB_MAXRECLENG - return the maximum number of element impedences defined for any element (max rec length)
  c output arguments:
  c eimiqr (int,func,out) - the returned value of eimiqr is based on setting of key.

7.8.29. eimget Function (Getting an Element Face Impedance)

*deck,eimget
  function eimget (ielem,iface,value)
  c *** primary function: get an element face impedance
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c ielem (int,sc,in) - element number
  c iface (int,sc,in) - face number (1-6)
  c output arguments:
  c eimget (int,func,out) - status of element.
  c =-1 - element has no impedances
  c = 0 - this element face has no impedances
  c > 0 - number of values defined
  c value (dp,ar(*),out) - the element impedances (real,imag)

7.8.30. eimput Subroutine (Storing an Element Impedance)

*deck,eimput
  subroutine eimput (ielem,iface,nval,value)
  c *** primary function: store an element face impedance.
  c *** Notice - This file contains ANSYS Confidential information ***
Chapter 7: Accessing the ANSYS Database

7.8.31. eimdel Subroutine (Deleting an Element Impedance)

*deck,eimdel
  subroutine eimdel (ielem,iface)
  c *** primary function: delete an impedance on a element
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c     ielem (int,sc,in) - element number
  c     iface (int,sc,in) - face number
  c                                      = 0 - delete all impedances on this
  c                                             element
  c                                      = 1-6 - delete impedance on this face
  c output arguments: none

7.8.32. esfiqr Function (Getting Information About Element Surface Stress Data)

*deck,esfiqr
  function esfiqr (ielem,key)
  c *** primary function: get information about element surface stress data
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c     ielem (int,sc,in) - element number (or zero, see below)
  c     key (int,sc,in)   - key as to the information needed
  c                 = 1 - return info about surface stress
  c                 ielem > 0 - return number of surface stresses on this
  c                                      element (rec length)
  c                 = 0 - return maximum number of surface stresses
  c                                      on any element (max rec length)
  c                 = DB_NUMDEFINED - return the number of surface stresses
  c                                      defined in model
  c output arguments:
  c     esfiqr (int,func,out) - the returned value of esfiqr is based on
  c                                      setting of key

7.8.33. esfget Function (Getting Element Surface Stress Data)

*deck,esfget
  function esfget (ielem,value)
  c *** primary function: get element surface stress data.
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c     ielem (int,sc,in) - element number
  c output arguments:
  c     esfget (int,func,out) - status of element.
  c                                      = 0 - element undefined
  c                                      > 0 - number of values returned
  c     value (dp,ar(*),out) - element surface stress data.
7.8.34. esfput Subroutine (Storing Element Surface Stress Data)

*deck,esfput
  subroutine esfput (ielem,nval,value)
  c *** primary function: store surface stresses for an element.
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  c     input arguments:
  c     ielem     (int,sc,in)       - element number
  c     nval      (int,sc,in)       - the total number of values
  c     value    (dp,ar(nval),in)   - the values
  c
  c     output arguments:  none

7.8.35. esfdel Subroutine (Deleting an Element's Surface Stress Data)

*deck,esfdel
  subroutine esfdel (ielem)
  c *** primary function: delete element surface stress data
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  c     input arguments:
  c     ielem    (int,sc,in)       - element number
  c                                     = 0 - delete for all defined elements
  c
  c     output arguments:  none.

7.8.36. efsdel Subroutine (Deleting a Flagged Surface on an Element)

*deck,efsdel
  subroutine efsdel (ielem,iface)
  c *** primary function: delete a flagged surface on an element
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  c     input arguments:
  c     ielem     (int,sc,in)       - element number
  c     iface     (int,sc,in)       - face number
  c                                     = 0 - all flagged surfaces
  c                                     = 1-6 - this flagged surface
  c
  c     output arguments:  none.

7.8.37. efsget function (Getting Element Face Flagged Surfaces)

*deck,efsget
  function efsget (ielem,iface,value)
  c *** primary function: get element face flagged surfaces
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  c     input arguments:
  c     ielem    (int,sc,in)       - element number
  c     iface    (int,sc,in)       - face number (1-6)
  c
  c     output arguments:
  c     efsget   (int,func,out)    - status of element.
  c                                     ==-1 - no values for this element
  c                                     = 0 - zero flagged surfaces defined
  c                                     > 0 - number of values defined
  c     value    (dp ,ar(*),out)   - the element flagged surfaces
7.8.38. efsiqr function (Getting Information About Flagged Surfaces)

```c
*deck,efsiqr
  function efsiqr (ielem,iface,key)
  c *** primary function: get information about flagged surfaces
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c     ielem   (int,sc,in)   - element number for inquire.
  c     iface   (int,sc,in)   - face number for inquire (0-6)
  c     key     (int,sc,in)   - key as to the information needed
  c       =  1       - return flagged surfaces mask for element
  c       =  5       - return number of flagged surfaces for this
  c           element face
  c     = DB_NUMDEFINED,
  c     = DB_MAXDEFINED - return value is based on setting of iface
  c           NOTE: both DB_NUMDEFINED and
  c           DB_MAXDEFINED produce the same
  c           functionality
  c     iface   =  0       - return total number of pressures,
  c           convections, etc defined in model
  c           = 1-6       - return number of flagged surfaces
  c           defined for this element. (rec length)
  c           NOTE: only 1-6 is valid, but this
  c           routine simply checks that iface is in
  c           the range. The actual value of iface
  c           does not matter in this case.
  c     = DB_MAXRECLENG - return maximum number of flagged surfaces
  c           for any element (max rec length)
  c output arguments:
  c     efsiqr   (int,func,out) - the returned value of efsiqr is based on
  c           setting of key.
```

7.8.39. efsput Subroutine (Storing an Element Face Flagged Surface)

```c
*deck,efsput
  subroutine efsput (ielem,iface,nval,value)
  c *** primary function: store an element face flagged surface.
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c     ielem   (int,sc,in)   - element number
  c     iface   (int,sc,in)   - face number (1-6)
  c     nval    (int,sc,in)   - number of values to put
  c     value   (dp ,ar(nval),in) - the element flagged surface values
  c output arguments:  none.
```

7.9. Results Information Routines

7.9.1. dspiqr Function (Getting Information About Nodal Results)

```c
*deck,dspiqr
  function dspiqr (node,key)
  c *** primary function: get information about nodal results
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
```
7.9.2. dspget Function (Getting a Nodal Result from the Database)

*deck,dspget
   function dspget (node,ndf,idf,value)
   c *** primary function:    get a nodal result from the data base
   c *** Notice - This file contains ANSYS Confidential information ***
   c typ=int,dp,log,chr,dcp siz=sc,ar(n),func intent=in,out,inout
   c input arguments:
   c variable (typ,siz,intent)     description
   c node     (int,sc,in)       - node number
   c ndf      (int,sc,in)       - number of results requested
   c idf      (int,ary(ndf),in) - reference number for the DOF: (1-32)
   c UX = 1, UY = 2, UZ = 3, ROTX= 4, ROTY= 5, ROTZ= 6, AX = 7, AY = 8
   c AZ = 9, VX =10, VY =11, VZ =12
   c PRES=19, TEMP=20, VOLT=21, MAG =22, ENKE=23, ENDS=24
   c EMF =25, CURR=26, SP01=27, SP02=28, SP03=29, SP04=30, SP05=31, SP06=32
   c (missing entries are spares)
   c output arguments:
   c dspget   (int,func,out)    - number of actual results
   c value    (dp,ar(ndf),out)  - result values

7.9.3. dspput Subroutine (Storing a Constraint at a Node)

*deck,dspput
   subroutine dspput (node,ndf,idf,value)
   c *** primary function:    store a result at a node.
   c *** Notice - This file contains ANSYS Confidential information ***
   c input arguments:
   c node     (int,sc,in)       - node number
   c ndf      (int,sc,in)       - number of results to be stored
   c idf      (int,ary(ndf),in) - reference number for the DOF: (1-32)
   c value    (dp,ar(ndf),in)   - displacement values
   c output arguments:  none

7.9.4. dspdel Subroutine (Deleting a Result at a Node)

*deck,dspdel
   subroutine dspdel (node,ndf,idf)
   c *** primary function:    delete a result at a node
   c *** Notice - This file contains ANSYS Confidential information ***
   c input arguments:
   c node     (int,sc,in)       - node number. (0 to delete DOF at all
   c nodes)
   c ndf      (int,sc,in)       - number of DOFs to delete (0 to delete
   c all DOFs)
   c idf      (int,ary(*),in)   - reference number for the DOF: (1-32)
   c UX = 1, UY = 2, UZ = 3, ROTX= 4, ROTY= 5, ROTZ= 6, AX = 7, AY = 8
   c AZ = 9, VX =10, VY =11, VZ =12
7.9.5. emsiqr Function (Getting Information About an Element's Miscellaneous Summable Data)

*deck, emsiqr
  function emsiqr (ielem,key)
  c *** primary function: get information about element misc summable data
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c    ielem (int,sc,in) - element number (or zero, see below)
  c    key (int,sc,in) - key as to the information needed
  c           = 1 - return info about misc summed data records
  c           ielem > 0 - return number of misc summed data items for this element
  c           (record length)
  c           = 0 - return maximum number of misc summed data items on any element (max record length)
  c           = DB_NUMDEFINED - return total number of misc summed data items defined in model
  c output arguments:
  c    emsiqr (int,func,out) - the returned value of emsiqr is based on setting of key

7.9.6. emsget Function (Getting an Element's Miscellaneous Summable Data)

*deck, emsget
  function emsget (ielem,value)
  c *** primary function: get element misc summable data.
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c    ielem (int,sc,in) - element number
  c output arguments:
  c    emsget (int,func,out) - status of element.
  c           = 0 - element is undefined
  c           > 0 - number of data items returned
  c    value (dp,ar(*),out) - element misc summed data.
  c NOTE: the contents of this record is element dependent. See SMISC on ETABLE command

7.9.7. emsput Subroutine (Storing an Element's Miscellaneous Summable Data)

*deck, emsput
  subroutine emsput (ielem,nval,value)
  c *** primary function: store misc. summable data for an element.
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c    ielem (int,sc,in) - element number
  c    nval (int,sc,in) - number of values to be stored
  c    value (dp,ar(nval),in) - the misc summed data values
  c output arguments: none
  c NOTE: the contents of this record is element dependent. See SMISC on ETABLE command
7.9.8. emsdel Subroutine (Deleting an Element's Miscellaneous Summable Data)

*deck, emsdel
  subroutine emsdel (ielem)
  c *** primary function: delete element misc summable data
  c *** Notice - This file contains ANSYS Confidential information ***
  c    input arguments:
  c    ielem (int, sc, in) - element number
  c                                     = 0 - delete data for all defined elements
  c    output arguments: none

7.9.9. enfiqr Function (Getting Information About Element Nodal Forces)

*deck, enfiqr
  function enfiqr (ielem, key)
  c *** primary function: get information about element nodal forces
  c *** Notice - This file contains ANSYS Confidential information ***
  c    input arguments:
  c    ielem (int, sc, in) - element number (or zero, see below)
  c    key (int, sc, in) - key as to the information needed
  c                     = 1 - return info about element nodal forces
  c                     ielem > 0 - return number of element nodal forces for this element
  c                     (record length)
  c                     = 0 - return maximum number of element nodal forces on any element
  c                     (max record length)
  c                     = DB_NUMDEFINED - return total number of element nodal forces defined in model
  c    output arguments:
  c    enfiqr (int, func, out) - the returned value of enfiqr is based on setting of key

7.9.10. enfget Function (Getting an Element's Nodal Forces)

*deck, enfget
  function enfget (ielem, value)
  c *** primary function: get element nodal forces.
  c *** Notice - This file contains ANSYS Confidential information ***
  c    input arguments:
  c    ielem (int, sc, in) - element number
  c    output arguments:
  c    enfget (int, func, out) - status of element.
  c                     = 0 - element has no nodal forces
  c                     > 0 - number of nodal forces returned
  c    value (dp, ar(*), out) - element nodal forces

7.9.11. enfput Subroutine (Storing an Element's Nodal Forces)

*deck, enfput
  subroutine enfput (ielem, nval, value)
  c *** primary function: store nodal force results at an element.
  c *** Notice - This file contains ANSYS Confidential information ***
  c    input arguments:
  c    ielem (int, sc, in) - element number
7.9.12. enfdel Subroutine (Deleting an Element's Nodal Forces)

*deck, enfdel
subroutine enfdel (ielem)
c *** primary function: delete element nodal forces data
c *** Notice - This file contains ANSYS Confidential information ***
c     input arguments:
c        ielem    (int,sc,in)       - element number
                                 = 0 - delete for all defined elements

c     output arguments:  none

7.9.13. ensiqr Function (Getting Information About an Element's Nodal Stresses)

*deck, ensiqr
function ensiqr (ielem,key)
c *** primary function: get information about element nodal stresses
c *** Notice - This file contains ANSYS Confidential information ***
c     input arguments:
c        ielem     (int,sc,in)      - element number (or zero, see below)
c        key     (int,sc,in)        - key as to the information needed
                                 = 1 - return info about element nodal stresses
                                 ielem > 0 - return number of element nodal
                                 stresses for this element (record length)
                                 = 0 - return maximum number of element
                                 nodal stresses on any element (max record length)
                                 = DB_NUMDEFINED - return total number of element
                                 nodal stresses defined in model

c     output arguments:
c        ensiqr   (int,func,out)    - the returned value of ensiqr is based on
                                       setting of key

7.9.14. ensget Function (Getting an Element's Nodal Stresses)

*deck, ensget
function ensget (ielem,value)
c *** primary function: get element nodal stresses.
c *** Notice - This file contains ANSYS Confidential information ***
c     input arguments:
c        ielem    (int,sc,in)       - element number

c     output arguments:
c        ensget   (int,func,out)    - status of element.
                                 = 0 - element undefined
                                 > 0 - number of nodal stresses returned

        value    (dp,ar(*),out)    - element nodal stresses
                                 NOTE: Stresses at each corner node in the order
                                 X, Y, Z, XY, YZ, XZ, S1, S2, S3, SI, SE
                                 For solid elements, stresses at each
For shell elements, stresses at each corner node (first top surface, then bottom).
For layered elements (w/KEYOPT(8)=0), stresses for "first" layer at each corner node (first at the bottom surface of the bottom layer, then the top surface of the top layer). Stresses for "second" layer at each corner node (first the bottom surface, then the top surface for the layer with the largest failure criteria). The second layer is not present if failure criteria were not used or are not appropriate.
For layered elements (w/KEYOPT(8)=1), stresses for each layer at each corner node (first at the bottom surface, then the top surface).
For beam elements, the contents of this record is element dependent. See LS item of ETABLE command.

7.9.15. ensput Subroutine (Storing Nodal Stresses at an Element)

*deck,ensput
  subroutine ensput (ielem,nval,value)
  c *** primary function: store nodal stresses at an element.
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c   ielem   (int,sc,in) - element number
  c   nval    (int,sc,in) - the total number of values
  c       (6*nnod*nface)
  c   value   (dp,ar(nval),in) - the stress values
  c output arguments: none

  NOTE: Stresses at each corner node in the order
       X, Y, Z, XY, YZ, XZ, S1, S2, S3, SI, SE
  For solid elements, stresses at each corner node
  For shell elements, stresses at each corner node (first top surface, then bottom)
  For layered elements (w/KEYOPT(8)=0), stresses for "first" layer at each corner node (first at the bottom surface of the bottom layer, then the top surface of the top layer). Stresses for "second" layer at each corner node (first the bottom surface, then the top surface for the layer with the largest failure criteria). The second layer is not present if failure criteria were not used or are not appropriate.
  For layered elements (w/KEYOPT(8)=1), stresses for each layer at each corner node (first at the bottom surface, then the top surface).
  For beam elements, the contents of this record is element dependent. See LS item of ETABLE command.
7.9.16. ensdel Subroutine (Deleting an Element's Nodal Stresses)

*deck, ensdel
subroutine ensdel (ielem)
c *** primary function: delete element nodal stresses
c *** Notice - This file contains ANSYS Confidential information ***
c input arguments:
c ielem (int, sc, in) - element number
c = 0 - delete for all defined elements
c output arguments: none.

7.9.17. engiqr Function (Getting Information About an Element's Energies)

*deck, engiqr
function engiqr (ielem, key)
c *** primary function: get information about element energies
c *** Notice - This file contains ANSYS Confidential information ***
c input arguments:
c ielem (int, sc, in) - element number (or zero, see below)
c key (int, sc, in) - key as to the information needed
  = 1 - return info about element energies
  = 0 - return number of element energies on
  this element (rec length)
  = DB_NUMDEFINED - return the number of element energies
  defined in model
output arguments:
c engiqr (int, func, out) - the returned value of engiqr is based on
  setting of key

7.9.18. engget Function (Getting an Element's Energies)

*deck, engget
function engget (ielem, value)
c *** primary function: get element energies.
c *** Notice - This file contains ANSYS Confidential information ***
c input arguments:
c ielem (int, sc, in) - element number
output arguments:
c engget (int, func, out) - status of element.
c = 0 - element undefined
c = 11 - energies returned
c value (dp, ar(6), out)
  value(1) = volume of element
  (2) = strain energy
  (3) = dissipation energy
  (4) = kinetic energy
  (5) = plastic energy
  (6) = creep energy
  (7) = stabilization energy
  (8) = spares
  (9) = thermal energy
  (10-11) = spares
7.9.19. engput Subroutine (Storing an Element's Energies and Volume)

*deck,engput
   subroutine engput (ielem,nval,value)
   c *** primary function: store volume and energies for an element.
   c *** Notice - This file contains ANSYS Confidential information ***
   c   input arguments:
   c   ielem     (int,sc,in)       - element number
   c   nval      (int,sc,in)       - the total number of values to be stored
   c   value    (dp,ar(6),in)      - volume and energies
   c                 value(1) = volume of element
   c                 (2) = strain energy
   c                 (3) = dissipation energy
   c                 (4) = kinetic energy
   c                 (5) = plastic energy
   c                 (6) = creep energy
   c                 (7) = stabilization energy
   c                 (8) = spares
   c                 (9) = thermal energy
   c                 (10-11) = spares
   c   output arguments: none

7.9.20. engdel Subroutine (Deleting an Element's Energies)

*deck,engdel
   subroutine engdel (ielem)
   c *** primary function: delete element energies
   c *** Notice - This file contains ANSYS Confidential information ***
   c   input arguments:
   c   ielem    (int,sc,in)       - element number
   c                     = 0 - delete for all defined elements
   c   output arguments: none.

7.9.21. egriqr Function (Getting Information About an Element's Nodal Gradients)

*deck,egriqr
   function egriqr (ielem,key)
   c *** primary function: get information about element nodal gradients
   c *** Notice - This file contains ANSYS Confidential information ***
   c   input arguments:
   c   ielem     (int,sc,in)      - element number (or zero, see below)
   c   key     (int,sc,in)        - key as to the information needed
   c        = 1 - return info about nodal gradients
   c        for ielem > 0 - return number of nodal gradients on
   c        this element (record length)
   c        = 0 - return maximum number of nodal
   c        gradients on any element
   c        (maximum record length)
   c        = DB_NUMDEFINED - return the number of nodal gradients defined
   c        in model
   c   output arguments:
   c   egriqr   (int,func,out)    - the returned value of egriqr is based on
   c        setting of key
7.9.22. egrget Function (Getting an Element's Nodal Gradients)

```
*deck, egrget

function egrget (ielem, value)

  c *** primary function: get element nodal gradients.

  c *** Notice - This file contains ANSYS Confidential information ***

  c input arguments:
  c    ielem (int, sc, in)    - element number

  c output arguments:
  c    egrget (int, func, out) - status of element.
  c                = 0 - element undefined
  c                > 0 - number of nodal gradients returned
  c    value (dp, ar(*), out) - element nodal gradients

Note: If a coupled field, a set of gradients are stored in the following order (as available): fluid, thermal, electric, magnetic
```

7.9.23. egrput Subroutine (Storing an Element's Nodal Gradients)

```
*deck, egrput

subroutine egrput (ielem, nval, value)

  c *** primary function: store nodal gradients at an element.

  c *** Notice - This file contains ANSYS Confidential information ***

  c input arguments:
  c    ielem (int, sc, in)    - element number
  c    nval (int, sc, in)    - the total number of values
  c    value (dp, ar(nval), in) - the gradient values

  c Note: If a coupled field, a set of gradients are stored in the following order (as appropriate): fluid, thermal, electric, magnetic

  c output arguments: none
```

7.9.24. egrdel Subroutine (Deleting an Element's Nodal Gradients)

```
*deck, egrdel

subroutine egrdel (ielem)

  c *** primary function: delete element nodal gradients

  c *** Notice - This file contains ANSYS Confidential information ***

  c input arguments:
  c    ielem (int, sc, in)    - element number
  c                = 0 - delete for all defined elements

  c output arguments: none.
```

7.9.25. eeliqr Function (Getting Information About an Element's Nodal Elastic Strains)

```
*deck, eeliqr

function eeliqr (ielem, key)

  c *** primary function: get information about element nodal elastic strains

  c *** Notice - This file contains ANSYS Confidential information ***

  c input arguments:
```
7.9.26. eelget Function (Getting an Element's Nodal Elastic Strains)

*deck,eelget

function eelget (ielem,value)

*** primary function: get element nodal elastic strains.

*** Notice - This file contains ANSYS Confidential information ***

input arguments:

ielem  (int,sc,in) - element number

output arguments:

eelget  (int,func,out) - status of element.

value   (dp,ar(*),out) - element nodal elastic strains

NOTE: Strains at each corner node in the order
X, Y, Z, XY, YZ, XZ
For solid elements, strains at each
corner node
For shell elements, strains at each
corner node (first top surface, then
bottom)
For layered elements (w/KEYOPT(8)=0),
strains for "first" layer at each
corner node (first at the bottom
surface of the bottom layer, then the
top surface of the top layer).
Strains for "second" layer at each
corner node (first the bottom surface,
then the top surface for the layer with
the largest failure criteria).
The second layer is not present if
failure criteria were not used or are
not appropriate
For layered elements (w/KEYOPT(8)=1),
strains for each layer at each
corner node (first at the bottom surface, then
the top surface)
For beam elements, the contents of this
record is element dependent. See LEPEL
item of ETABLE command.

7.9.27. eelput Subroutine (Storing an Element's Nodal Elastic Strains)

*deck,eelput

subroutine eelput (iel,ival,ival)

*** primary function: store nodal elastic strains at an element.

*** Notice - This file contains ANSYS Confidential information ***
Chapter 7: Accessing the ANSYS Database

7.9.28. **eeldel Subroutine (Deleting an Element's Nodal Elastic Strains)**

*deck,eeldel
  subroutine eeldel (ielem)
  c *** primary function: delete element elastic strains
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c     ielem  (int,sc,in) - element number
  c     = 0 - delete for all defined elements
  c output arguments: none.

7.9.29. **epliqr Function (Getting Information About an Element's Nodal Plastic Strains)**

*deck,epliqr
  function epliqr (ielem,key)
  c *** primary function: get information about element nodal plastic strains
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c     ielem  (int,sc,in) - element number (or zero, see below)
  c     key   (int,sc,in) - key as to the information needed
  c        = 1 - return info about plastic strains
  c        = 0 - return maximum number of nodal plastic strains
  c        = DB_NUMDEFINED - return the number of nodal plastic strains defined in model
  c output arguments:
7.9.30. eplget Function (Getting an Element’s Nodal Plastic Strains)

*deck,eplget
function eplget (ielem,value)
c *** primary function:    get element nodal plastic strains.
c *** Notice - This file contains ANSYS Confidential information ***
c     input arguments:
c        ielem    (int,sc,in)       - element number

c     output arguments:
c        eplget   (int,func,out)    - status of element.
c                                   = 0 - element undefined
                                   > 0 - number of nodal plastic strains returned
        value    (dp,ar(*),out)    - element nodal plastic strains

NOTE: Strains at each corner node in the order
      X, Y, Z, XY, YZ, XZ
For solid elements, strains at each corner node
For shell elements, strains at each corner node (first top surface, then bottom)
For layered elements (w/KEYOPT(8)=0), strains for "first" layer at each corner node (first at the bottom surface of the bottom layer, then the top surface of the top layer).
Strains for "second" layer at each corner node (first the bottom surface, then the top surface for the layer with the largest failure criteria).
The second layer is not present if failure criteria were not used or are not appropriate
For layered elements (w/KEYOPT(8)=1), strains for each layer at each corner node (first at the bottom surface, then the top surface)
For beam elements, the contents of this record is element dependent. See LEPPL item of ETABLE command.

7.9.31. eplput Subroutine (Storing an Element’s Nodal Plastic Strains)

*deck,eplput
subroutine eplput (ielem,nval,value)
c *** primary function:    store nodal plastic strains at a element.
c *** Notice - This file contains ANSYS Confidential information ***
c     input arguments:
c        ielem     (int,sc,in)       - element number
        nval      (int,sc,in)       - the total number of values
                                   (6*nnod*nface)
        value    (dp,ar(nval),in)   - the strain values

c     output arguments:  none

NOTE: Strains at each corner node in the order
      X, Y, Z, XY, YZ, XZ
For solid elements, strains at each corner node
For shell elements, strains at each corner node (first top surface, then bottom).

For layered elements (w/KEYOPT(8)=0), strains for "first" layer at each corner node (first at the bottom surface of the bottom layer, then the top surface of the top layer).

Strains for "second" layer at each corner node (first the bottom surface, then the top surface for the layer with the largest failure criteria). The second layer is not present if failure criteria were not used or are not appropriate.

For layered elements (w/KEYOPT(8)=1), strains for each layer at each corner node (first at the bottom surface, then the top surface).

For beam elements, the contents of this record is element dependent. See LEPPL item of ETABLE command.

7.9.32. epldel Subroutine (Deleting an Element's Nodal Plastic Strains)

*deck, epldel
  subroutine epldel (ielem)
  c *** primary function: delete element plastic strains
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c        ielem    (int, sc, in)       - element number
  c                                     = 0 - delete for all defined elements
  c output arguments:  none.

7.9.33. ecriqr Function (Getting Information About an Element's Nodal Creep Strains)

*deck, ecriqr
  function ecriqr (ielem, key)
  c *** primary function: get information about element nodal creep strains
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c        ielem     (int, sc, in)      - element number (or zero, see below)
  c        key     (int, sc, in)        - key as to the information needed
    = 1 - return info about creep strains
    ielem > 0 - return number of nodal creep strains
    on this element
    (record length)
    = 0 - return maximum number of nodal creep strains on any element
    (max record length)
    = DB_NUMDEFINED - return the number of nodal creep strains
    defined in model
  c output arguments:
  c        ecriqr   (int, func, out)    - the returned value of ecriqr is based on
                                      setting of key

7.9.34. ecrget Function (Getting an Element's Nodal Creep Strains)

*deck, ecrget
  function ecrget (ielem, value)
  c *** primary function: get element nodal creep strains.
**7.9.35. ecrput Subroutine (Storing an Element's Nodal Creep Strains)**

*deck,ecrput

subroutine ecrput (ielem,nval,value)

c *** primary function: store nodal creep strains at an element.

c *** Notice - This file contains ANSYS Confidential information ***

c input arguments:
  c ielem (int,sc,in) - element number
  c nval (int,sc,in) - the total number of values
  c (6*nmod*nface)
  c value (dp,ar(nval),in) - the strain values

c output arguments: none

NOTE: Strains at each corner node in the order
  X, Y, Z, XY, YZ, XZ
  For solid elements, strains at each corner node
  For shell elements, strains at each corner node (first top surface, then bottom)
  For layered elements (w/KEYOPT(8)=0), strains for "first" layer at each corner node (first at the bottom surface of the bottom layer, then the top surface of the top layer).
  Strains for "second" layer at each corner node (first the bottom surface, then the top surface for the layer with the largest failure criteria).
  The second layer is not present if failure criteria were not used or are not appropriate
  For layered elements (w/KEYOPT(8)=1), strains for each layer at each corner node (first at the bottom surface, then the top surface)
  For beam elements, the contents of this record is element dependent. See LEFCD item of ETABLE command.
failure criteria were not used or are not appropriate
For layered elements (w/KEYOPT(8)=1), strains for each layer at each corner node (first at the bottom surface, then the top surface)
For beam elements, the contents of this record is element dependent. See LEPCR item of ETABLE command.

7.9.36. ecrdel Subroutine (Deleting an Element's Nodal Creep Strains)

*deck,ecrdel
    subroutine ecrdel (ielem)
c *** primary function: delete element creep strains
c *** Notice - This file contains ANSYS Confidential information ***

    input arguments:
    ielem  (int,sc,in) - element number
           = 0 - delete for all defined elements

    output arguments: none.

7.9.37. ethiqr Function (Getting Information About an Element's Nodal Thermal Strains)

*deck,ethiqr
    function ethiqr (ielem,key)
c *** primary function: get information about element nodal thermal strains
c *** Notice - This file contains ANSYS Confidential information ***

    input arguments:
    ielem     (int,sc,in) - element number (or zero, see below)
    key     (int,sc,in) - key as to the information needed
         =  1 - return info about thermal strains
               on this element
         = 0 - return maximum number of nodal thermal strains
               on any element
         = DB_NUMDEFINED - return the number of nodal thermal strains
               defined in model

    output arguments:
    ethiqr   (int,sc,out) - the returned value of ethiqr is based on setting of key

7.9.38. ethget Function (Getting an Element's Nodal Thermal Stresses)

*deck,ethget
    function ethget (ielem,value)
c *** primary function: get element nodal thermal strains. also the volumetric swelling strain
c *** Notice - This file contains ANSYS Confidential information ***

    input arguments:
    ielem    (int,sc,in) - element number

    output arguments:
    ethget   (int,func,out) - status of element.
         = 0 - element undefined
         > 0 - number of nodal thermal strains returned
    value    (dp,ar(*),out) - element nodal thermal strains

    NOTE: Strains at each corner node in the order...
7.9.39. ethput Subroutine (Storing an Element's Nodal Thermal Stresses)

*deck, ethput
   subroutine ethput (ielem, nval, value)
   c *** primary function: store nodal thermal strains at an element.
   c also the volumetric swelling strain
   c *** Notice - This file contains ANSYS Confidential information ***
   c
   input arguments:
   c ielem    (int, sc, in) - element number
   c nval     (int, sc, in) - the total number of values
   c          (6*nnod*nface)
   c value    (dp, ar(nval), in) - the strain values
   c
   output arguments: none
   NOTE: Strains at each corner node in the order
   X, Y, Z, XY, YZ, XZ, epswel
   For solid elements, strains at each corner node
   For shell elements, strains at each corner node (first at the bottom)
   For layered elements (w/KEYOPT(8)=0),
   strains for "first" layer at each corner node (first at the bottom surface of the bottom layer, then the top surface of the top layer).
   Strains for "second" layer at each corner node (first the bottom surface, then the top surface for the layer with the largest failure criteria).
   The second layer is not present if failure criteria were not used or are not appropriate
   For layered elements (w/KEYOPT(8)=1),
   strains for each layer at each corner node (first at the bottom surface, then the top surface)
   For beam elements, the contents of this record is element dependent. See LEPTH item of ETABLE command.
7.9.40. ethdel Subroutine (Deleting an Element's Thermal, Initial, and Swelling Strains)

*deck, ethdel
  subroutine ethdel (ielem)
  c *** primary function: delete element thermal, initial, and
  c                      swelling strains
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  input arguments:
  c       ielem    (int,sc,in)   - element number
  c          = 0 - delete for all defined elements
  c
  output arguments: none.

7.9.41. euliqr Function (Getting Information About an Element's Euler Angles)

*deck, euliqr
  function euliqr (ielem,key)
  c *** primary function: get information about element euler angles
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  input arguments:
  c       ielem     (int,sc,in)   - element number (or zero, see below)
  c       key       (int,sc,in)   - key as to the information needed
  c           =  1 - return info about element euler angles
  c           ielem > 0 - return number of euler angles on this
  c                       element
  c           (record length)
  c           = 0 - return maximum number of euler angles
  c                       on any element
  c           (max record length)
  c           = DB_NUMDEFINED - return the number of element euler angles
  c                       defined in model
  c
  output arguments:
  c       euliqr   (int,func,out) - the returned value of euliqr is based on
  c                       setting of key

7.9.42. eulget Function (Getting an Element's Nodal Euler Angles)

*deck, eulget
  function eulget (ielem,value)
  c *** primary function: get element nodal euler angles.
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  input arguments:
  c       ielem    (int,sc,in)   - element number
  c
  output arguments:
  c       eulget   (int,func,out) - status of element.
  c           = 0 - element undefined
  c           > 0 - number of euler angle values
  c                       returned
  c       value    (dp,ar(*),out) - element euler angles
  c
  NOTE: For lower-ordered elements, rotations
  c      at centroid
  c      For higher-ordered elements, rotations
  c      at each corner node
  c      For layered shells, rotations at each
  c      corner node, plus layer rotation angle
  c      for each layer (real constant THETA)
  c      For layered solids, rotation angles at
  c      centroid, plus layer rotation angle
  c      for each layer (real constant THETA)
7.9.43. eulput Subroutine (Storing an Element's Euler Angles)

*deck,eulput
  subroutine eulput (ielem,nval,value)
c *** primary function: store nodal euler angles for an element.
c *** Notice - This file contains ANSYS Confidential information ***
c
  input arguments:
c    ielem    (int,sc,in)   - element number
c    nval     (int,sc,in)   - the total number of values
  (3 * number of display nodes)
c    value    (dp,ar(nval),in) - the euler angle values

c  output arguments:  none

NOTE: For lower-ordered elements, rotations at centroid
For higher-ordered elements, rotations at each corner node
For layered shells, rotations at each corner node, plus layer rotation angle
for each layer (real constant THETA)
For layered solids, rotation angles at centroid, plus layer rotation angle
for each layer (real constant THETA)

7.9.44. euldel Subroutine (Deleting an Element's Euler Angles)

*deck,euldel
  subroutine euldel (ielem)
c *** primary function: delete element euler angles
  (or zero, see below)
c *** Notice - This file contains ANSYS Confidential information ***
c
  input arguments:
c    ielem    (int,sc,in)   - element number
  = 0 - delete for all defined elements

c  output arguments:  none.

7.9.45. efxiqr Function (Getting Information About Element Fluxes)

*deck,efxiqr
  function efxiqr (ielem,key)
c *** primary function: get information about element fluxes
  (record length)
c *** Notice - This file contains ANSYS Confidential information ***
c
  input arguments:
c    ielem    (int,sc,in) - element number (or zero, see below)
c    key       (int,sc,in) - key as to the information needed
    = 1 - return info about element fluxes
  ielem > 0 - return number of fluxes on this element
  (record length)
c    = 0 - return maximum number of fluxes
  on any element
  (max record length)
c    = DB_NUMDEFINED - return the number of element fluxes defined
  in model

c  output arguments:
c    efxiqr   (int,func,out) - the returned value of efxiqr is based on
  setting of key
7.9.46. efxget Function (Getting an Element Flux)

```c
*deck,efxget
  function efxget (ielem,value)
c *** primary function: get element nodal fluxes.
c *** Notice - This file contains ANSYS Confidential information ***
c  input arguments:
c    ielem    (int,sc,in)       - element number

c  output arguments:
c    efxget   (int,func,out)    - status of element.
c                              = 0 - element undefined
nc                              > 0 - number of nodal fluxes returned
nc    value    (dp,ar(*),out)    - element nodal fluxes
nc Note: If a coupled field, a set of fluxes is stored in the following order (as available): fluid, thermal, electric, magnetic
```

7.9.47. efxput Subroutine (Storing an Element's Fluxes)

```c
*deck,efxput
  subroutine efxput (ielem,nval,value)
c *** primary function: store nodal fluxes at an element.
c *** Notice - This file contains ANSYS Confidential information ***
c  input arguments:
c    ielem     (int,sc,in)       - element number
nc    nval      (int,sc,in)       - the total number of values (ndir*nnod*nscailr)
c    value    (dp,ar(nval),in)   - the flux values

c  output arguments: none
nc Note: If a coupled field, a set of fluxes is stored in the following order (as available): fluid, thermal, electric, magnetic
```

7.9.48. efxdel Subroutine (Deleting Element Fluxes)

```c
*deck,efxdel
  subroutine efxdel (ielem)
c *** primary function: delete element nodal fluxes
nc *** Notice - This file contains ANSYS Confidential information ***
c  input arguments:
c    ielem    (int,sc,in)       - element number
nc                              = 0 - delete for all defined elements

c  output arguments: none.
```

7.9.49. elfiqr Function (Getting Information About Element Local Forces)

```c
*deck,elfiqr
  function elfiqr (ielem,key)
c *** primary function: get information about elem local forces
nc *** Notice - This file contains ANSYS Confidential information ***
c  input arguments:
c    ielem    (int,sc,in)       - element number (or zero, see below)
```
### 7.9.50. elfget Function (Getting an Element Local Force)

*deck, elfget

function elfget (ielem, value)

c *** primary function: get element local nodal forces.

c *** Notice - This file contains ANSYS Confidential information ***

input arguments:
- `ielem` (int, sc, in) - element number

output arguments:
- `elfget` (int, func, out) - status of element.
- `value` (dp, ar(*), out) - element local nodal forces.

### 7.9.51. elfput Subroutine (Storing an Element's Local Forces)

*deck, elfput

subroutine elfput (ielem, nval, value)

```c
variable (typ, siz, intent) description
ielem (int, sc, in) - element number
nval (int, sc, in) - the total number of values
value (dp, ar(nval), in) - element local nodal forces
```

output arguments: none

### 7.9.52. elfdel Subroutine (Deleting Element Local Forces)

*deck, elfdel

subroutine elfdel (ielem)

```c
input arguments:
ielem (int, sc, in) - element number
= 0 - delete for all defined elements
```

output arguments: none.
7.9.53. emnqr Function (Getting Information About Element Miscellaneous Non-summable Data)

*deck, emnqr
  function emnqr (ielem, key)
  c *** primary function: get information about element misc non-summable data
  c
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  input arguments:
  c    ielem (int, sc, in) - element number (or zero, see below)
  c    key  (int, sc, in) - key as to the information needed
  c          = 1 - return info about element misc non-summed data
  c          ielem > 0 - return number of data items on this element
  c          (record length)
  c          = 0 - return maximum number of data items on any element
  c          (max record length)
  c          = DB_NUMDEFINED - return the number of element misc non-summed data items defined in model
  c
  output arguments:
  c    emnqr (int, func, out) - the returned value of emnqr is based on setting of key
  c
7.9.54. emnget Function (Getting an Element's Miscellaneous Non-summable Data)

*deck, emnget
  function emnget (ielem, value)
  c *** primary function: get misc non-summable data.
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  input arguments:
  c    ielem (int, sc, in) - element number
  c
  output arguments:
  c    emnget (int, func, out) - status of element.
  c          = 0 - no non-summed misc data at this element
  c          > 0 - number of data items returned
  c    value (dp, ar(*), out) - element misc non-summable data.
  c
  NOTE: the contents of this record is element dependent. See NMISC on ETABLE command

7.9.55. emnput Subroutine (Storing an Element's Miscellaneous Non-summable Data)

*deck, emnput
  subroutine emnput (ielem, nval, value)
  c *** primary function: store misc. non-summable data for an element.
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  input arguments:
  c    ielem  (int, sc, in) - element number
  c    nval   (int, sc, in) - the total number of values
  c    value  (dp, ar(nval), in) - the misc. non-summable data items
  c
  output arguments: none
  c
  NOTE: the contents of this record is element dependent. See NMISC on ETABLE command
7.9.56. emndel Subroutine (Deleting an Element's Miscellaneous Non-summable Data)

*deck,emndel
  subroutine emndel (ielem)
  c *** primary function: delete element misc non-summable data
  c *** Notice - This file contains ANSYS Confidential information ***
  c     input arguments:
  c        ielem    (int,sc,in)       - element number
  c                                     = 0 - delete for all defined elements
  c     output arguments:  none.

7.9.57. ecdiqr Function (Getting Information About Element Current Densities)

*deck,ecdiqr
  function ecdiqr (ielem,key)
  c *** primary function: get information about element current densities
  c *** Notice - This file contains ANSYS Confidential information ***
  c     input arguments:
  c        ielem     (int,sc,in)      - element number (or zero, see below)
  c        key       (int,sc,in)      - key as to the information needed
  c                 =  1 - return info about element current densities
  c                             ielem > 0 - return number of current densities on
  c                                           this element
  c                                     (record length)
  c                 = 0 - return maximum number of current
  c                                           densities on any element
  c                                     (max record length)
  c                 = DB_NUMDEFINED - return the number of element current
  c                                    densities defined in model
  c     output arguments:
  c        ecdiqr   (int,func,out)    - the returned value of ecdiqr is based on
  c                                       setting of key

7.9.58. ecdget Function (Getting an Element Current Density)

*deck,ecdget
  function ecdget (ielem,value)
  c *** primary function: get calculated element current densities.
  c *** Notice - This file contains ANSYS Confidential information ***
  c     input arguments:
  c        ielem    (int,sc,in)       - element number
  c     output arguments:
  c        ecdget   (int,func,out)    - status of element.
  c                                     = 0 - element has no current densities
  c                                     > 0 - number of calculated element
  c        value    (dp,ar(*),out)    - calculated element current densities.
  c                                      NOTE: current densities are in the order
  c                                       X, Y, Z

7.9.59. ecdput Subroutine (Storing an Element's Current Densities)

*deck,ecdput
  subroutine ecdput (ielem,nval,value)
  c *** primary function: store calculated element current densities
7.9.60. ecddel Subroutine (Deleting Element Current Densities)

*deck, ecddel
subroutine ecddel (ielem)
c *** primary function: delete element current densities

7.9.61. enliqr Function (Getting Information About Element Nonlinear Tables)

*deck, enliqr
function enliqr (ielem, key)
c *** primary function: get information about element nonlinear tables

7.9.62. enlget Function (Getting Element Nonlinear Tables)

*deck, enlget
function enlget (ielem, value)
c *** primary function: get element nonlinear tables.
7.9.63. enlput Subroutine (Storing an Element’s Nonlinear Tables)

*deck,enlput
  subroutine enlput (ielem,n,temp)
  c *** primary function:    store element nonlinear tables
  c *** Notice - This file contains ANSYS Confidential information ***
  c     input arguments:
  c        ielem    (int,sc,in)       - element number
  c        n        (int,sc,in)       - number of element nonlinear table values
  c        temp     (dp ,ar(6),in)    - element nonlinear table,etc.
  c     output arguments:  none.
  c                                   NOTE: Nonlinear data at each node are in the
  c                                          order SEPL, SRAT, HPRES, EPEQ, PSV,
  c                                          PLWK, and 4 spares
  c                                         For beam elements, the contents and
  c                                          number of information is element
  c                                          dependent.  See NLIN on ETABLE
  c                                          command

7.9.64. enldel Subroutine (Deleting Element Nonlinear Tables)

*deck,enldel
  subroutine enldel (ielem)
  c *** primary function:    delete element nonlinear tables
  c *** Notice - This file contains ANSYS Confidential information ***
  c     input arguments:
  c        ielem    (int,sc,in)       - element number
  c                                     = 0 - delete for all defined elements
  c     output arguments:  none.

7.9.65. ehciqr Function (Getting Information About Calculated Element Heat Generations)

*deck,ehciqr
  function ehciqr (ielem,key)
  c *** primary function: get information about calculated elem heat generations
  c *** Notice - This file contains ANSYS Confidential information ***
  c     input arguments:
  c        ielem     (int,sc,in)      - element number (or zero, see below)
  c        key       (int,sc,in)      - key as to the information needed
  c                 =  1 - return info about calculated element heat gens
  c                         for ielem > 0 - return number of heat gens for
  c                         this element
  c                 =  0 - return maximum number of heat gens
  c                 = DB_NUMDEFINED - return the number of calculated element heat
  c                                    generations defined in model
  c     output arguments:
  c        ehciqr   (int,func,out)    - the returned value of ehciqr is based on
  c                     setting of key
7.9.66. ehcget Function (Getting a Calculated Element Heat Generation)

*deck,ehcget
  function ehcget (ielem,value)
  c *** primary function: get calculated element heat generations.
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  input arguments:
  c     ielem  (int,sc,in)  - element number
  c
  output arguments:
  c     ehcget  (int,func,out)  - status of element.
  c        = 0 - element undefined
  c        > 0 - number of calculated element heat generations
  c
  value  (dp,ar(*),out)  - calculated element heat generations.

7.9.67. ehcput Subroutine (Storing an Element's Calculated Heat Generations)

*deck,ehcput
  subroutine ehcput (ielem,nval,value)
  c *** primary function: store calculated element heat generations
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  input arguments:
  c     ielem  (int,sc,in)  - element number
  c
  nval  (int,sc,in)  - the total number of values
  c
  value  (dp,ar(nval),in)  - calculated element heat generations.
  c
  output arguments: none.

7.9.68. ehcdel Subroutine (Deleting Element Calculated Heat Generations)

*deck,ehcdel
  subroutine ehcdel (ielem)
  c *** primary function: delete calculated element heat generations
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  input arguments:
  c     ielem  (int,sc,in)  - element number
  c        = 0 - delete for all defined elements
  c
  output arguments: none.
Chapter 8: Subroutines for Users' Convenience

This chapter describes ANSYS routines available to you for use in programming. Using these routines isn't required, but may make your life easier. These routines include a set of general routines that perform utility-type functions, a set of routines supporting vector functions, a set of routines supporting matrix functions, and routines supporting message processing options.

8.1. Input and Output Abbreviations

The descriptions of inputs and outputs for the routines discussed in this chapter use the following abbreviations:

- **Argument type** is one of the following:
  - int - integer
  - dp - double precision
  - log - logical
  - chr - character
  - dcp - double precision complex

- **Argument size** is one of the following:
  - sc - scalar variable
  - ar(n) - array variable of length n
  - func - functional return value

- **Argument intent** is one of the following:
  - in - input argument
  - out - output argument
  - inout - both an input and an output argument

8.2. General Subroutines

8.2.1. dptoch Subroutine (Retrieve Eight Characters From a Double Precision Variable)

```c
*deck,dptoch
subroutine dptoch (dp8,ch8)
c *** primary function: retreive 8 characters from a dp variable
c *** Notice - This file contains ANSYS Confidential information ***
c !!! NOTICE to programmers: this routine does not convert from a !!!
c !!! machine-independent format! Use dpexttoch if this dp word !!!
c !!! came from a common or non-char database record !!!
c input arguments:
c dp8     (dp,sc,in)  - dp variable containing characters
c output arguments:
c ch8     (ch*8,sc,out) - characters retreived from the dp word
```

8.2.2. wrinqr Function (Obtain Information About Output)

```c
*deck,wrinqr
function wrinqr (key)
c *** primary function: obtain information about output
c *** Notice - This file contains ANSYS Confidential information ***
```
Chapter 8: Subroutines for Users' Convenience

--- caution: the following variables are "saved/resumed".
--- key=WR_COLINTER thru WR_SUPCOLMAX in "wrinqr/wrinfo"
--- (data for "/fmt,/page,/header" commands).
--- note that the whole common cannot be "saved/resumed". cwa

typ=int,dp,log,chr,dcp   siz=sc,ar(n),func   intent=in,out,inout

input arguments:
variable (typ,siz,intent)    description                        wrcom name

  key      (int,sc,in)

  = WR_PRINT       - print flag (kprint)                        prtkey
     wrinqr = 0 - no output
      = 1 - print
  = WR_OUTPUT      - current output unit number (iott)         outfit
  = WR_MASTEROUT   - master output file                        frstot
  = WR_COLINTER   - interactive columns per page               intcol
  = WR_COLBATCH   - batch columns per page                      batcol
  = WR_LINEINTER  - interactive lines per page                 lintlin
  = WR_LINEBATCH  - batch lines per page                        batlin
  = WR_CHARITEM   - characters per output item                 chrper
  = WR_CHARDECIMAL - characters past decimal                    chrddec
  = WR_CHARINTEGER - characters in leading integer             chrint
  = WR_CHARTYPE   -                                          chrtyp
     wrinqr = 1 - using E format in output
                  = 2 - using F format in output
                  = 3 - using G format in output
  = WR_SUPTITLE   - tlabel supress key                         keyhed
  = WR_SUPSUBTITLE - subtitle supress key                      keyit
  = WR_SUPSILTER   - ls,iter id supress key                    keyid
  = WR_NOTELINE    - note line supress key                     keynot
  = WR_SUPCOLHEADER - column header supress key                keylab
  = WR_SUPCOLMAX   - column maximum supress key                keysum
  = WR_LISTOPT     - ListOpt from /output command              ListOpt

output arguments:
  wrinqr   (int,func,out)      - the value corresponding to key

8.2.3. erinqr Subroutine (Obtaining Information from the Errors Common)

*deck,erinqr
  function erinqr (key)

  *** primary function:    obtain information from errors common

  *** Notice - This file contains ANSYS Confidential information ***

input arguments:
  key      (int,sc,in)

     = item to be returned

     1=keyerr, 2=errfil, 3=numnot, 4=numwrn,
     5=numerr, 6=numfat, 7=maxmsg, 8=lvler,
     9=mpcmand, 10=nercmd, 11=nterim, 12=nomore,
     13=creopen, 14=kaerr, 15=kystat, 16=muxr45,
     17=mshkey, 19=opterr, 20=flown,
     21=errhp, 22=oneport, 23=psderr, 24=mpcmw
     25=kystop, 26=icloads, 27=ifkey

  ---- below definitions copied from ercomm 7/92 for user information

     *** key number= ........................
     (see ansysdef for parameter definitions)
     |
     \/

co keyerr - master error flag
co errfil - errors file unit number
co numnot - total number of notes displayed
co numwrn - total number of warnings displayed
co numerr - total number of errors displayed
co numfat - total number of fatals displayed
co maxmsg - max allowed number of displayed messages before abort
co lvler - used basics in solution (from cnvr command.)
co -1=do not set keyerr for notes/errors/warnings.

Programmer's Manual for ANSYS . ANSYS Release 11.0 . 002328 . © SASIP, Inc. All rights reserved.
Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

260
8.2.4. TrackBegin Subroutine (Beginning Tracking for a Subroutine Call)

*deck,TrackBegin
  subroutine TrackBegin (sub32)

  c *****function: mark beginning of track ansys call
  c *** Notice - This file contains ANSYS Confidential information ***
8.2.5. TrackEnd Subroutine (Ending Tracking for a Subroutine Call)

```fortran
*deck, TrackEnd
subroutine TrackEnd (sub32)
```

**Input Arguments:**
- `sub32` (char*(*) sc in) - name of subroutine being left (32 characters max)

**Output Arguments:** None

8.2.6. erhandler Subroutine (Displaying ANSYS Errors)

```fortran
*deck, erhandler
subroutine erhandler (filein,msgid,msglvl,lngstrng,dperr,cherr)
```

**Primary Function:** Display ANSYS error messages

**Input Arguments:**
- `filein` (ch*40 sc in) - Filename used for character portion of message ID (this is the file name of the file which contains the source for this routine)
  - if 'ErrorMessageProbe', then error was generated on another processor (distributed ANSYS). In that case, dperr contains the message already made ASCII and expanded
- `msgid` (int sc in) - Numeric portion of the message ID 1 - 9999, unique for each erhandler call in the FILE. Recommend using a sequence, similar to format conventions, i.e., 5000, 5010, 5020
  - if filein='ErrorMessageProbe', this is the CPU # that originally generated the error
- `msglvl` (int sc in) - level of error (same as lngerr)
  - 0 = no label (used for u/i pop-ups)
  - -1 = no label (used for u/i pop-ups) timed as a note message
  - 1 = note, 2 = warning, 3 = error, 4 = fatal
  - -3 = error w/tech supp note
  - -4 = fatal w/tech supp note
  - (see lngerr.F for text of tech supp note)
- `lngstrng` (ch*(*) sc in) - error message to display. Use keywords of %i %g %c %/ for formatting (same as lngerr)
- `dperr` (dp, ar(*), in) - vector of data to display. Contains both integer and double precision data. (same as lngerr)
  - if filein='ErrorMessageProbe', dperr contains the unpacked message and lngstrng and cherr are ignored
- `cherr` (ch*(*) ar(*), in) - vector of character data to display max length of character data is 32 characters
8.2.7. intrp Subroutine (Doing Single Interpolation)

*deck,intrp
  subroutine intrp (klog,kppx,kstpz,xval,ax,ay,yval,nmax,kyoff)
  c *** primary function: **** subroutine for single interpolation ****
  c (if double interpolation is needed, see intrpt)
  c
  c *** Notice - This file contains ANSYS Confidential information ***
  c
typ=int,dp,log,chr,dcp  siz=sc,ar(n),func     intent=in,out,inout
  c
  c input arguments:
  c variable (typ,siz,intent)      description
  c klog     (int,sc,in)       - interpolation type
  c - 0 - use linear interpolation
  c - 1 - use log-log interpolation
  c -- note: there is no option yet for
  c lin-log or log-lin
  c kppx     (int,sc,in)       - X value end of table signal
  c - 0 - a repeated x-value will signal the end
  c of the table
  c - 1 - a repeated x-value will not signal the end of the table
  c (only known use = c evaluation)
  c kstpz    (int,sc,in)       - Y value end of table signal
  c - 0 - a yval of zero will not signal the end
  c of the table (e.g. stress fitting)
  c - 1 - a yval of zero will signal the end of the table
  c (in general, material properties (exception: alpx))
  c
  c NOTE: the end of the table will be signaled thru
  c either of the above conditions, or more
  c commonly, that nmax values have been processed,
  c or that the present x table entry is less than
  c the previous one (ax(i) .lt. ax(i-1)).
  c evaluations done after the end of the table are
  c evaluated as if they were at the end of the
  c table. similarly, evaluations done before the
  c beginning of the table are done as if they were
  c done at the beginning of the table.
  c
  c xval     (dp,sc,in)         - value of x with which to go into the table
  c ax       (dp,ar(*) ,in)     - table of x values, in ascending order
  c ay       (dp,ar(*) ,in)     - table of y values
  c nmax     (int,sc,in)        - maximum table size allowed
  c
  c output arguments:
  c yval     (dp,sc,out)        - value of y which comes back from the table
  c kyoff    (int,sc,out)       - xval status flag
  c - 0 - xval in x range
  c - 1 - xval less than minimum x
  c - 2 - xval greater than maximum x

8.2.8. tranx3 Subroutine (Processing Geometry for 3-D Line Elements)

*deck,tranx3
  subroutine tranx3 (nnod,xyz,nx,tr)
  c *** primary function: geometric processor for 3-d line elements
  c with or without a 3rd node
  c
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  c input arguments:
  c nnod    (int,sc,in)       - number of nodes (2 or 3)
  c xyz     (dp,ar(nx,*) ,in) - coordinates (x,y,z down)
  c nx      (int,sc,in)       - row dimension of xyz array
  c
8.2.9. systop Subroutine (Stopping an ANSYS Program Run)

```
*deck,systop
  subroutine systop (icode)
  c *** primary function: stop an ansys run
  c *** secondary functions: pass an error code to the system
  c
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  c input arguments:
  c    icode    (int,sc,in)       - stop error code (0<icode<127)
  c                                   0 - normal exit
  c                                   1 - stack overflow error
  c                                   2 - stack level overflow
  c                                   3 - stack pop below zero
  c                                   4 - names do not match in stkpxp
  c                                   5 - command line argument error
  c                                   6 - unused (was: accounting file error)
  c                                   7 - licensing failure
  c                                   8 - indicated error or end-of-run
  c                                   11 - error in user routine
  c                                   12 - macro stop command
  c                                   13 - job already running
  c                                   14 - untrapped xox error
  c                                   15 - anserr fatal error
  c                                   16 - possible full disk
  c                                   17 - possible corrupted or missing file
  c                                   18 - Error in VM routines (corrupt db?)
  c                                   21 - unauthorized code section entered
  c                                   25 - unable to open x11 server
  c                                   30 - quit signal
  c                                   31 - failure to get signal in max time
  c                                   (syhold)
  c                                   >32 - system dependent error
  c                                   35 - fatal error on another process
  c                                  (distributed ANSYS)
  c
  c output arguments:  none
```

8.3. Vector Functions

8.3.1. vdot Function (Computing the Dot Product of Two Vectors)

```
*deck,vdot
  function vdot (v1,v2,n)
  c *** primary function: compute dot product of vectors v1 and v2
  c
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  c input arguments:
  c    v1       (dp,ar(n),in)     - vector v1
  c    v2       (dp,ar(n),in)     - vector v2
  c    n        (int,sc,in)       - length of vectors v1 and v2
  c
  c output arguments:
  c    vdot     (dp,sc,out)       - dot product of v1 and v2
  c
```

8.3.2. vsum Function (Summing Vector Components)

```
*deck,vsum
  function vsum (va,n)
  c *** primary function: sum the components of a vector
```
**8.3.3. vmax Function (Retrieving the Maximum Vector Value at a Given Location)**

```c
*deck,vmax
function vmax (v,n,locmax)
c *** primary function: return the max value and location in a vector
c *** Notice - This file contains ANSYS Confidential information ***
c input arguments:
  c  v    (dp,ar(n),in)  - vector v
  c  n    (int,sc,in)    - length of vector v
c output arguments:
  c  locmax  (int,sc,out) - location of max value in vector v
  c  vmax    (dp,sc,out)  - max value in vector v
```

**8.3.4. lastv Function (Retrieving the Position of the Last Nonzero Term in a Double Precision Vector)**

```c
*deck,lastv
function lastv (v,n)
c ********* find position of last non-zero term in a d.p. vector *********
```

**8.3.5. izero Function (Setting an Integer Vector to Zero)**

```c
*deck,izer
subroutine izero (ivect,n)
c ********** set an integer vector to zero **********
c
```

**8.3.6. imove Function (Assigning Equal Values to Two Integer Vectors)**

```c
*deck,imove
subroutine imove (i1,i2,n)
c ********** move a vector from one to another **********
c
```

**8.3.7. vzero Subroutine (Initializing a Vector to Zero)**

```c
*deck,vzero
subroutine vzero (v,n)
c *** primary function: initialize a vector to zero
c *** Notice - This file contains ANSYS Confidential information ***
c input arguments:
  c  v    (dp,ar(n),inout)  - vector to be zeroed out
  c  n    (int,sc,in)    - number of words to zero out
c output arguments:
  c  v    (dp,ar(n),inout)  - zeroed vector
```
8.3.8. vmove Subroutine (Moving One Vector into Another)

```fortran
*deck,vmove
  subroutine vmove (v1,v2,n)
  c *** primary function: copy a vector into another vector
  c
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  c *** Note: This function can have unpredictable results if v1 and v2
  c           reference the same array and overlap.
  c
  c input arguments:
  c     v1       (dp,ar(*),in)     - vector v1
  c     n        (int,sc,in)       - length of vectors v1, v2
  c
  c output arguments:
  c     v2       (dp,ar(*),out)    - vector v2
  c
```

8.3.9. vimove Subroutine (Moving One Vector into Another Incrementally)

```fortran
*deck,vimove
  subroutine vimove (v1,inc1,v2,inc2,n)
  c *** primary function: move one vector into another
  c
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  c input arguments:
  c     v1       (dp,ar(inc1,n),in)  - vector v1
  c     inc1     (int,sc,in)         - increment on vector v1
  c     inc2     (int,sc,in)         - increment on vector v2
  c     n        (int,sc,in)         - number of items to be moved
  c
  c output arguments:
  c     v2       (dp,ar(inc2,n),in)  - vector v2
  c
```

8.3.10. vinit Subroutine (Assigning a Scalar Constant to a Vector)

```fortran
*deck,vinit
  subroutine vinit (v,n,const)
  c *** primary function: initialize a vector to a constant
  c
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  c input arguments:
  c     n        (int,sc,in)       - length of vector v
  c     const    (dp,sc,in)        - constant to set vector v to
  c
  c output arguments:
  c     v        (dp,ar(n),out)    - vector v
  c
```

8.3.11. viinit Subroutine (Assigning a Scalar Constant to a Vector Incrementally)

```fortran
*deck,viinit
  subroutine viinit (v,inc,n,const)
  c *** primary function: set the components of vector v to const by increments
  c
  c *** Notice - This file contains ANSYS Confidential information ***
  c
  c input arguments:
  c     inc      (int,sc,in)         - increment (first dimension) of vector v
  c     n        (int,sc,in)         - length (second dimension) of vector v
  c     const    (dp,sc,in)          - constant to set components of vector v to
  c
```
8.3.12. vapb Subroutine (Setting a Vector to Sum of Two Vectors)

*deck, vapb
subroutine vapb (a,b,c,n)
c *** primary function: add vector a to vector b to get vector c
c *** Notice - This file contains ANSYS Confidential information ***

c input arguments:
  c a (dp,ar(n),in) - a vector
  c b (dp,ar(n),in) - b vector
  c n (int,sc,in) - length of vectors a,b,c

c output arguments:
  c c (dp,ar(n),out) - c vector

c
8.3.13. vapb1 Subroutine (Combining Two Vectors in One)

*deck, vapb1
subroutine vapb1 (a,b,n)
c *** primary function: add vector b to vector a, and store in vector a

8.3.14. vapcb1 Subroutine (Multiplying a Vector to a Constant)

*deck, vapcb1
subroutine vapcb1 (a,b,n,const)
c *** primary function: multiply vector b to constant, add to vector a, and store in vector a

8.3.15. vamb Subroutine ( Gets a Third Vector by Subtracting One Vector from Another)

*deck, vamb
subroutine vamb (a,b,c,n)
c *** primary function: subtract vector b from vector a to get vector c
c *** Notice - This file contains ANSYS Confidential information ***

c input arguments:
  c a (dp,ar(n),in) - vector a
  c b (dp,ar(n),in) - vector b
  c n (int,sc,in) - length of vectors a,b,c

c output arguments:
  c c (dp,ar(n),out) - vector c

c
8.3.16. vamb1 Subroutine (Subtracting One Vector from Another)

*deck, vamb1
subroutine vamb1 (a,b,n)
c *** primary function: subtract vector b from vector a and save in vector a
c *** Notice - This file contains ANSYS Confidential information ***

c input arguments:
  c a (dp,ar(n),inout) - a vector
  c b (dp,ar(n),in) - b vector
  c n (int,sc,in) - length of vectors a,b

Programmer's Manual for ANSYS . ANSYS Release 11.0 . 002328 . © SAS IP, Inc. All rights reserved.
Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.
8.3.17. vmult Subroutine (Multiplying a Vector by a Constant)

```fortran
*deck, vmult
    subroutine vmult (v1, v2, n, const)
    c *** primary function: multiply a vector by a constant
    c *** Notice - This file contains ANSYS Confidential information ***
    c
    c input arguments:
    c     v1       (dp, ar(n), in)     - vector v1
    c     n        (int, sc, in)       - length of vectors v1, v2
    c     const    (dp, sc, in)        - constant to multiply v1
    c
    c output arguments:
    c     v2       (dp, ar(n), out)    - vector v2
```

8.3.18. vmult1 Subroutine (Multiplying a Vector by a Constant)

```fortran
*deck, vmult1
    subroutine vmult1 (v1, n, const)
    c *** primary function: multiply a vector by a constant
    c *** Notice - This file contains ANSYS Confidential information ***
    c
    c input arguments:
    c     v1       (dp, ar(n), inout)  - vector v1
    c     n        (int, sc, in)       - length of vector n
    c     const    (dp, sc, in)        - constant to multiply v1
    c
    c output arguments:
    c     v1       (dp, ar(n), inout)  - vector v1
```

8.3.19. vcross Subroutine (Defining a Vector via a Cross Product)

```fortran
*deck, vcross
    subroutine vcross (a, b, c)
    c primary function: calculate c = a x b
    c *** Notice - This file contains ANSYS Confidential information ***
    c     typ=int, dp, log, chr, dcp  siz=sc, ar(n)  intent=in, out, inout
    c
    c input arguments:
    c     a        (dp, ar(3), in)     - first vector to be cross-multiplied
    c     b        (dp, ar(3), in)     - second vector to be cross-multiplied
    c
    c output arguments:
    c     c        (dp, ar(3), out)    - resulting vector
```

8.3.20. vnorme Subroutine (Normalizing a Three-Component Vector)

```fortran
*deck, vnorme
    subroutine vnorme (iel, v)
    c primary function: normalize a vector to unit length
    c *** Notice - This file contains ANSYS Confidential information ***
    c
    c input arguments:
    c     iel     (int, sc, inout)    - element number
    c
    c output arguments:
```
c if the vector length is zero.

c *** Notice - This file contains ANSYS Confidential information ***

c input arguments:
c    iel      (int,sc,inout) - element number
c    v        (dp,ar(3),inout) - vector to be normalized

c output arguments:
c    iel      (int,sc,inout) - if 0, vector has zero length
c    v        (dp,ar(3),inout) - normalized vector

8.3.21. vnorm Subroutine (Normalizing a Vector to Unit Length)

*deck,vnorm
  subroutine vnorm (v,n)
  c *** primary function: normalize a vector to unit length
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c    v        (dp,ar(n),inout) - vector v
  c    n        (int,sc,inout) - dimension length of vector v
  c output arguments:
  c    v        (dp,ar(n),inout) - normalized vector v
  c    n        (int,sc,inout) - n = 0 if error in operation

8.3.22. ndgxyz Function (Getting the X,Y,Z Vector for a Node)

*deck,ndgxyz
  function ndgxyz (node,xyz)
  c *** primary function: get x,y,z vector for a node.
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c    node     (int,sc,in) - node number for operation.
  c output arguments:
  c    ndgxyz   (int,sc,out) - status of node.
  c    xyz      (dp,ar(3),out) - vector containing x,y,z

8.3.23. ndpxyz Subroutine (Storing X,Y,Z for a Node)

*deck,ndpxyz
  subroutine ndpxyz (node,xyz)
  c *** primary function: store x,y,z vector for a node.
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c    node     (int,sc,in) - node number for operation.
  c    xyz      (dp,ar(3),in) - vector containing x,y,z (vector should be in global system)
  c output arguments: none
8.4. Matrix Subroutines

8.4.1. maxv Subroutine (Multiplying a Vector by a Matrix)

*deck,maxv
subroutine maxv (a,v,w, nr,nc)
c *** primary function: multiply a matrix by a vector

*** Notice – This file contains ANSYS Confidential information ***

input arguments:
c a (dp,ar(nr,*),in) - matrix a
c v (dp,ar(*),in) - vector v
c nr (int,sc,in) - number of rows in matrix a
c nc (int,sc,in) - number of columns to multiply in matrix a

output arguments:
c w (dp,ar(*),out) - product vector w

8.4.2. maxv1 Subroutine (Multiplying a Vector by a Matrix)

*deck,maxv1
subroutine maxv1 (a,v, nr,nc)
c *** primary function: multiply a vector by a matrix

*** Notice – This file contains ANSYS Confidential information ***

input arguments:
c a (dp,ar(nr,nc),in) - matrix a
c v (dp,ar(nc),inout) - vector v
c nr (int,sc,in) - number of rows in matrix a
*** nr limited to 60 ***
c nc (int,sc,in) - number of columns to multiply in matrix a

output arguments:
c v (dp,ar(nr),inout) - product, stored in vector v

8.4.3. matxv Subroutine (Multiplying a Vector by a Full Transposed Matrix)

*deck,matxv
subroutine matxv (a,v,w, nr,nc)
c *** primary function: multiply vector by full transposed matrix

*** Notice – This file contains ANSYS Confidential information ***

input arguments:
c a (dp,ar(nr,*),in) - matrix a (first dimension must = nr)
c v (dp,ar(nv),in) - vector v (nv must be greater or equal to nr)
c nr (int,sc,in) - first dimension and number of active rows of the untransposed matrix a
c c (also the number of active rows of vector v)
c c nc (int,sc,in) - number of columns of the untransposed matrix a
(also the number of computed items in the product vector w)
c c if negative, accumulate

output arguments:
c w (dp,ar(na,*),out) - product vector w
8.4.4. matxv1 Subroutine (Multiplying a Vector by a Full Transposed Matrix)

*deck,matxv1
  subroutine matxv1 (a,v,nr,nc)
  c *** primary function: multiply vector by full transposed matrix
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c a (dp,ar(nr,*),in) - matrix a
  c v (dp,ar(nr),inout) - vector v
  c nr (int,sc,in) - number of rows in matrix (un-transposed)
  c nc (int,sc,in) - number of columns in matrix (un-transposed)
  c *** nc limited to 60 ***
  c output arguments:
  c v (dp,ar(nc),inout) - product, stored in vector v

8.4.5. matxb Subroutine (Transposing a matrix)

*deck,matxb
  subroutine matxb (a,b,c, na,nb,nc, n1,n2,n3)
  c *** primary function: (a)t * (b) = (c)   t means transpose
  c *** Notice - This file contains ANSYS Confidential information ***
  c input arguments:
  c a (dp,ar(na,*),in) - matrix a
  c b (dp,ar(nb,*),in) - matrix b
  c na (int,sc,in) - number of rows in matrix a
  c nb (int,sc,in) - number of rows in matrix b
  c nc (int,sc,in) - number of rows in matrix c
  c n1 (int,sc,in) - number of rows in matrix c to fill
  c n2 (int,sc,in) - number of columns in matrix c to fill
  c n3 (int,sc,in) - number of columns in matrix a and
  c number of rows of matrix b
  c to work with (the two need
  c to be the same for the inner product)
  c if n3 is negative, accumulate results in c
  c output arguments:
  c c (dp,ar(nc,*),out) - product matrix c

8.4.6. maat Subroutine (Changing a Matrix Value via Addition, Multiplication, and Transposition)

*deck,maat
  subroutine maat(a,c, nc,n, con)
  c primary function: does con*a*at and sums the result onto c (a is a vector)
  c *** Notice - This file contains ANSYS Confidential information ***
  c typ=int,dp,log,chr,dcp siz=sc,ar(n) intent=in,out,inout
  c input arguments:
  c a (dp,ar(*),in) - vector to be multiplied by itself to
  c generate an nxn square matrix
  c a by a-transposed)
  c c (dp,ar(nc,*),inout) - matrix to be accumulated onto
  c nc (int,sc,in) - number of rows in the c matrix
  c n (int,sc,in) - size of square matrix
  c con (dp,sc,in) - multiplier on above square matrix
  c output arguments:
  c c (dp,ar(nc,*),inout) - matrix to be accumulated onto
8.4.7. matsym Subroutine (Filling the Upper Triangle from the Lower Triangle)

*cdeck,matsym
subroutine matsym (a,nd,n)
c primary function: fill upper triangle from lower triangle

c *** Notice - This file contains ANSYS Confidential information ***
c
typ=int,dp,log,chr,dcp siz=sc,ar(n) intent=in,out,inout

c input arguments:
c a   (dp,ar(nd,*),inout) - matrix to have its lower triangular part
c copied to its upper triangular part
c nd  (int,sc,in)         - number of rows of the a matrix
c n   (int,sc,in)         - size of matrix to be processed

c output arguments:
c a   (dp,ar(nd,*),inout) - matrix that has its lower triangular part
    copied to its upper triangular part

c
8.4.8. mctac Subroutine (Transposing a symmetric matrix)

*cdeck,mctac
subroutine mctac (a,na,c,nc,nold,nnew)
c *** Notice - This file contains ANSYS Confidential information ***

c input arguments:
c a   (dp,ar(na,na),inout) matrix to be pre and post multiplied
    (part operated on must be square(nold x nold) and symmetric)
c na  (int,sc,in)          first dimension of the a matrix
c c   (dp,ar(nc,nnew),in)  matrix to pre and post multiply a by
    (part used may be rectangular(nold x nnew))
c nc  (int,sc,in)          first dimension of the c matrix
c nold (int,sc,in)         size of part of 'A' matrix that is
to be processed(input size). maximum = 64
c nnew (int,sc,in)         size of part of 'A' matrix that
    results from this operation(output size).
c    maximum = 64

c output arguments:
c a   (dp,ar(na,na),inout) resulting matrix
    (still square(nnew x nnew) and symmetric).

c
8.4.9. tran Subroutine (Transposing a matrix)

*cdeck,tran
subroutine tran (zs,tr,nz,ntr,nrow,irot)
c primary function: perform   tr-transpose * zs * tr ************

c *** Notice - This file contains ANSYS Confidential information ***

c input arguments:
c variable (typ,siz,intent)    description
    zs   (dp,ar(nz,nz),inout) - matrix to be transformed
c tr   (dp,ar(ntr,ntr),in)   - transformation matrix
c nz   (int,sc,in)          - dimensioned size of zs matrix
c ntr  (int,sc,in)          - dimensioned size of tr matrix
**8.4.10. symeqn Subroutine (Solving Simultaneous Linear Equations)**

*deck,symeqn

    function symeqn (a,nd,n,nc,defFlag)

    c
    c primary function: solve a set of simultaneous linear equations
    c
    c secondary functions: invert a matrix
    c
    c          NOTE: this routine assumes that the matrix to be solved or
    c                 inverted is positive or negative definite. This routine
    c                 also assumes that the diagonals are all non-zero. If
    c                 this assumption is not true, use isimeq.F.
    c
    c *** Notice - This file contains ANSYS Confidential information ***
    c
    c input arguments:
    c
    c    variable (typ,siz,intent)    description
    c    a        (dp,ar(nd,*),inout) - matrix to be solved or inverted
    c                                      second dimension must be at least:
    c                                       n + abs(nc)
    c    nd       (int,sc,in)         - first dimension of the a matrix
    c    n        (int,sc,in)         - number of equations
    c    nc       (int,sc,in)         - number of additional columns.
    c                   if nc = -n or -n, invert n x n matrix and
    c                   put result in the n+1 to 2xn columns.
    c                   if nc is 0 or negative, nc will be reset to
    c                   n and then symeqn will set up identity
    c                   matrix after the input matrix, where the
    c                   result of the inversion will be put.
    c                   if nc is positive and less than n, do a
    c                   partial inversion. see example 1 below.
    c    defFlag  (int,sc,in)         - flag indicating that incoming matrix MUST be:
    c                                      -1 - negative definite
    c                                      0 - positive or negative definite
    c                                      1 - positive definite
    c
    c output arguments:
    c
    c    variable (typ,siz,intent)    description
    c    symeqn   (in,sc,out)         - 0 - non-singular matrix
    c                                      1 - singular matrix
    c                                      2 - near-singular matrix
    c    a        (dp,ar(nd,*),inout) - results or inverted matrix.
    c                                      starts in column n+1.
    c                                      note: original information is destroyed.
    c
    c example 1:   Solve three simultaneous linear equations:
    c    i = symeqn (a(1,1),3,3,1)
    c    calling routine has a dimensioned as a(3,4)
    c    each equation has its 3 coefficients in the first 3 columns,
    c    and the constant term is in the fourth column.
    c    solution is in fourth column.
    c
    c example 2:   Invert a 3x3 matrix:
    c    i = symeqn (a(1,1),3,3,-3)
    c    calling routine has a dimensioned as a(3,6)
    c    input matrix was input in first 3 columns
    c    output matrix in outup in last 3 columns
Appendix A. Creating External Commands in UNIX

External commands allow you to add your own customized extensions to ANSYS without relinking the program. You can create custom routines in C that access any of the ANSYS API functions, link them into shared libraries using the supplied utilities, and execute the routines via the "external command" feature within ANSYS. In addition, ANSYS provides special commands that list all available external commands and allow you to reset all currently referenced external commands.

External command capability is supported on all UNIX platforms. Refer to your installation guide for currently supported compilers; the following instructions assume the presence of compatible compilers and linkers.

A.1. Tasks in Creating an External Command

To create a functional external command, you will need to complete the following general steps:

• Create compilable source code.
• Create a shared library. This is facilitated by the gen_share utility and your system's make capability.
• Create an external table file (ans_ext.tbl), listing the various shared libraries, functions, and the related command.
• Set an environment variable pointing to the directory that holds the external table file.

The following sections detail each of these tasks.

A.1.1. Creating Compatible Code

You can create your functions using any of the API functions described in //ansys_inc/v110/ansys/custom/include/cAnsInterface.h, cAnsQuery.h, and cAnsPick.h. The following code segment demonstrates, at a minimal level, how to create functions that can be used as an entry point into a custom coded shared library.

The most important point in the following example it that the C program interface is an integer function that has one argument (a char pointer).

```c
#include "cAnsInterface.h"
#include "CAnsQuery.h"
/*
   --------------------------  Function Description ---------------------
   extfnc
   int extfnc(uecmd)
   char *uecmd;

   Purpose:
   Demonstrate C API entry function for an external command.

   Parameters:
   Input
   --------------------------
   uecmd
   The ANSYS external command string.

   Output
   --------------------------

   Return Value:
   The return value is ignored by the calling function;
```
Appendix A. Creating External Commands in UNIX

--- Function Description ---
*
int extfnc(char* ucmd)
{
    /* Note: ucmd is the entire command given to invoke this function */
    char* cmdsend = {"/COM, COMMAND SENT FROM EXTERNAL COMMAND"};
    char* querystr = {"NODE,,NUM,MAX"};
    char strtrtn[32];
    int i, itype;
    double dbltrtn;

    /* Send a simple command to be executed */
    i = cAnsSendCommand(cmdsend);

    /* Perform a simple query */
    i = cAnsGetValue(querystr,dbltrtn,strtrtn,itype);

    /* Display the value retrieved */
    cAnsPrintf("Max Node Number = %g\n",dbltrtn);

    return (i);

A.1.2. Creating a Shared Library

Once you have written the source code for your functions, you can create a Makefile (using the gen_share utility) to build a shared library. The utility creates the Makefile in the current directory. The Makefile incorporates all the interdependencies of the C source files it encounters in that current directory. The gen_share utility is meant to setup the basic build. The user may need to make modifications to the Makefile depending on the situation.

The gen_share utility has the following syntax:

    gen_share [-h] [-64] shared_object_name

where

-h
    Produces command help.

-64
    Configures the Makefile to use the -mips4 option for IRIX64.

shared_object_name
    Is the name that will be given to the shared library.

As gen_share is executing, you may see one or more "No match" messages. This is normal. The script is searching for .c, .f, and .F file types in the current directory and returns this message if it cannot locate any files matching one of those types.

To create a shared library called mylibrary.so, you would issue the following command:

    % gen_share mylibrary.so

The utility will produce a Makefile in the current directory. You will be able to generate the shared library by issuing the following command:

    make

For example, to create the shared library for mylibrary.so, you would issue the following command:

    % make
You will then find the specified shared library file in the current directory. You may also see warnings from the
make process, and you may need to modify the Makefile or your source code.

**A.1.3. Creating an External Table File**

The external table file (ans_ext.tbl) can reside in any directory (but you must specify that directory in the
**ANSYS_EXTERNAL_PATH** environment variable). The file contains an entry for each shared library function you
wish to allow ANSYS to access. There is no limit to the number of entries. The file entries have the following
format:

```
/shared/library/path/library.so ~cm_name function_name
```

where

```
/shared/library/path/library.so
```

Is the path to the directory that contains the shared library file. (Remotely mounted file systems are not recom-
manded.)

```
~cm_name
```

Is the command used to invoke the function within ANSYS. The command name must begin with a tilde (~) and
each command name must be unique within the first four characters. The command name must be eight char-
acters or less, including the tilde (~).

```
function_name
```

Is the name of the function that is referenced by the specified command name. (This must be unique within the
first four characters if multiple external commands are specified.)

For example, the following entry references the `/home/mydir/mylib/myobject.so` shared library and
the `myobject_` function. It specifies `~myobj` as the related command:

```
/home/mydir/mylib/myobject.so ~myobj myobject_
```

ANSYS also makes use of external commands, and places its own shared libraries and the associated external
table file in the `/ansys_inc/v110/ansys/lib/<platform>` directory (where `<platform>` is the directory
specific to your computing platform, such as `/sgi64` or `/hppa8000-64`).

ANSYS loads external commands in the following order:

- ANSYS first checks the `ans_ext.tbl` file in the `/ansys_inc/v110/ansys/lib/<platform>`
directory and loads any external commands referenced there.
- ANSYS then loads external commands referenced by the external table file in the directory designated
with the **ANSYS_EXTERNAL_PATH** environment variable (see section Section A.2.4: Setting the **ANSYS_EXTERNAL_PATH** Environment Variable).

If you designate a command name that has the same first four characters as a command listed in the `/an-
sys_inc/v110/ansys/lib/<platform>/ans_ext.tbl` file, you will not be able to access your command.
Therefore, it is a good practice to check the ANSYS external table file to make sure you have no external command
name conflicts. Do not modify the `/ansys_inc/v110/ansys/lib/<platform>/ans_ext.tbl` file. You
can also use the `~DEBUG` command to verify that no external command name conflicts exist.
Note

The shared library must be consistent with the computer type and OS level on which ANSYS will be executed.

A.1.4. Setting the ANSYS_EXTERNAL_PATH Environment Variable

Before launching ANSYS, you must first set the ANSYS_EXTERNAL_PATH to point to the directory containing the external table file. (For convenience, if you distribute your new functionality to other users they should set their .login or .cshrc files so that it is persistent from session to session.) For example, the following sets the environment variable to point to the /home/mydir directory.

    setenv ANSYS_EXTERNAL_PATH /home/mydir

A.1.5. Using External Commands

To call an external command, enter it as you would any other ANSYS command. You can also call external commands through either an APDL macro or UIDL script.

Note

Avoid recursive external commands; that is, avoid situations where an external command calls another external command.

A.1.6. Checking External Command Status

You can check what shared libraries are currently accessible by entering the ~DEBUG command in the command input window. The following figure shows an example of ~DEBUG command output.

    External Command Mappings:
    Command    Library                                 Function   Accessed?
    ~---------~---------~---------~---------~---------~---------~---------~
    ~excmd    /home/mydir/mycode/mycommand.so      excmd            YES

In this example, the output lists the command, the related shared library, the function, and if the command has been accessed.

A.1.7. Resetting External Commands

You can

- Close all shared libraries
- Free memory associated with external commands

by issuing the ~RESET command. The command issues the following message to confirm that the reset operation was complete.

    ~RESET was processed: The external command buffers have been cleared.

Note

The /CLEAR command also closes/resets all external command shared libraries.
Appendix B. Creating External Commands in Windows

This section describes the steps required to create external commands on Windows platforms.

B.1. Tasks in Creating an External Command

To create a functional external command, you will need to complete the following general steps:

- Create compatible C source code.
- Create an external definition file (projname.def).
- Create a new project in Microsoft Developer Studio.
- Create a shared library.
- Create an external table file (ans_ext.tbl), listing the various shared libraries, each function and the related command.
- Set the ANSYS_EXTERNAL_PATH environment variable

The following sections detail each of these tasks.

B.1.1. Creating Compatible Code

You can create your functions using any of the API functions described in `Program Files\Ansys Inc\V110\custom\include\cAnsysInterface.h`, `cAnsQuery.h`, and `cAnspick.h`. You can then execute these functions via the “external command” feature within ANSYS. In addition, ANSYS provides special commands that list all available external commands and allow you to reset all currently referenced external commands. The following code segment demonstrates, at a minimal level, how to create functions that can be used as an entry point into a custom coded shared library.

The most important point in the following example is:

- The C program interface is an integer function that has one argument (a char pointer).

```c
#include <windows.h>
#include "cAnsysInterface.h"
#include "CAnsQuery.h"

/*
 * ------------------------------ Function Description ------------------------------
 * extfnc
 * int extfnc(uecmd)
 * char *uecmd;
 *
 * Purpose:
 * Demonstrate C API entry function for an external command.
 *
 * Parameters:
 * Input
 * ------------------------------
 * uecmd
 * The ANSYS external command string.
 *
 * Output
 * ------------------------------
 */
```

The following code segment demonstrates, at a minimal level, how to create functions that can be used as an entry point into a custom coded shared library.
B.1.2. Creating a New Project

To build and link your code, you will first need to create a project in the Microsoft Developer Studio.

- After launching Developer Studio, press Ctrl+N (or choose New from the File menu). In the New window, click on the Projects tab. Select Win 32 Dynamic-Link Library, and specify your project name and directory.

- In the Workspace frame, click on the File View tab. Right click on the sub category projname files and press F or select Add Files to Project. Choose your source and definition files, ansys.lib, cAnsInterface.h and any other necessary header files. The import library file ansys.lib is in /ansys110/lib and the header file cAnsInterface.h is in /ansys110/extra/custom/include.

B.1.3. Creating an External Definition File

For each external command, you must declare it in the external definition file. The naming convention for this file is the name of your project with the .def extension; it must be located in your project directory. This file consists of the word EXPORTS on the first line, and the name(s) of the functions to be exported on each successive line. For the example function above:

```
EXPORTS
extfunc
```

B.1.4. Creating a Shared Library

Once all of the necessary files have been incorporated into your project, simply compile (Ctrl+F7) and build (F7) the project. In your project directory, Developer Studio will create a Debug directory and will place the library in that directory (projname.dll).

B.1.5. Creating an External Table File

The external table file (ans_ext.tbl) can reside in any directory (but you must specify that directory in the ANSYS_EXTERNAL_PATH environment variable). The file contains an entry for each shared library function you wish ANSYS to access. There is no limit to the number of entries. The file entries have the following format:

```
Programmer's Manual for ANSYS . ANSYS Release 11.0 . 002328 . © SAS IP, Inc. All rights reserved.
Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.
```
C:\shared\library\path\projname.dll ~cm_name function_name

where:

C:\shared\library\path\projname.dll is the path to the directory that contains the shared library file. (Remotely mounted file systems are not recommended.)

~cm_name is the command used to invoke the function within ANSYS. The command name must begin with a tilde (~) and the first four characters of each command name must be unique.

function_name is the name of the function that is referenced by the specified command name. (This must be unique within the first four characters if multiple external commands are specified.)

For example, the following entry references the C:\home\mydir\mylibs\myobject.dll shared library and the myobject function, and specifies ~myobj as the related command:

C:\home\mydir\mylibs\myobject.dll ~myobj myobject

ANSYS also makes use of external commands, and places its own shared libraries and the associated external table file in the C:\Program Files\Ansys Inc\V110\lib\platform directory (where platform is the directory specific to your computing platform, such as Intel). ANSYS loads external commands in the following order:

- ANSYS first checks the ans_ext.tbl file in the C:\Program Files\Ansys Inc\V110\lib\platform directory and loads any external commands referenced there.
- ANSYS then loads external commands referenced by the external table file in the directory designated with the ANSYS_EXTERNAL_PATH environment variable (see Section B.1.6: Setting the ANSYS_EXTERNAL_PATH Environment Variable).

If you designate a command name that has the same first four characters as a command listed in the C:\Program Files\Ansys Inc\V110\lib\platform file, you will not be able to access your command. Therefore, it is a good practice to check the ANSYS external table file to make sure you have no external command name conflicts. Do not modify the C:\Program Files\Ansys Inc\V110\lib\platform\ans_ext.tbl file. You can also use the ~DEBUG command to verify that no external command name conflicts exist.

**Note**

The shared library must be consistent with the computer type and OS level on which ANSYS will be executed.

### B.1.6. Setting the ANSYS_EXTERNAL_PATH Environment Variable

Before launching ANSYS, you must first set the ANSYS_EXTERNAL_PATH to point to the directory containing the external table file. In Windows NT, the environment variables are in System Properties, which can be accessed through the Control Panel. For example, the following string sets the environment variable to point to the C:\home\mydir directory.

`set ANSYS_EXTERNAL_PATH=C:\home\mydir`

### B.1.7. Using External Commands

To call an external command, enter it as you would any other ANSYS command in the ANSYS command window. You can also call external commands through either an APDL macro or UIDL routine.
Note

Avoid recursive external commands; that is, avoid situations where an external command calls another external command.

B.1.8. Checking External Command Status

You can check what shared libraries are currently accessible by entering the ~DEBUG command in the command input window. The following figure shows an example of ~DEBUG command output.

External Command Mappings:

<table>
<thead>
<tr>
<th>Command</th>
<th>Library</th>
<th>Function</th>
<th>Accessed?</th>
</tr>
</thead>
<tbody>
<tr>
<td>~excmd</td>
<td>/home/mydir/mycode/mycommand.so</td>
<td>excmd</td>
<td>YES</td>
</tr>
</tbody>
</table>

Note that the output lists the command, the related shared library, the function, and whether or not the command has been accessed.

B.1.9. Resetting External Commands

You can

- Close all shared libraries
- Free memory associated with external commands

by issuing the ~RESET command. This command issues the following message to confirm that the reset operation is complete.

~RESET was processed: The external command buffers have been cleared.

Note

The /CLEAR command also closes/resets all external command shared libraries.
Index

Symbols
3-D line elements, 263
3-D nodal coordinates, 144

A
Abbreviations
file access routines, 55
acelcm.inc file, 104
activating UPFs, 107
ANSYS
customized version of, 105
executing command from a user routine, 192
linking
optimization program to, 185
UPFs to, 105
parameters, 104
running as a subroutine, 174
standard file header, 57
start-up
accessing program logic at, 183
stopping, 264
tailoring with UPFs, 101
UPFs for, 101
ansyscust file, 106
ansysdef.inc file, 104
arc-length debugging, 109
auto time stepping debugging, 109
/AUX2 command, 55
AUX2 process, 55

B
basic element pass debugging, 110
Beam cross sections
reading, 85
Binary files
accessing, 53
characteristics of, 54
closing or deleting a blocked binary file, 60
Converting Two Integers into a Pointer, 61
copying contents, 62
opening a blocked binary file, 57
viewing contents, 55
bincl subroutine, 60
binhed subroutine, 58
binini subroutine, 55
biniqr function, 56
binlib library, 53
binrd8 subroutine, 59
binset function, 57
bintfo subroutine, 57
bintr subroutine, 62
bintst program, 61
bintwr subroutine, 62
binwrt8 subroutine, 59
Buffered binary I/O systems
initializing, 55
Buffered file
reading data from, 59
writing data to, 59

C
C, 101, 103
FORTRAN shell for, 103
.c extension, 105
calculating
rotation caused by internal pressure, 150
Calculating
dot product of two vectors, 264
sum of vector components, 264
CDREAD
using the command, 71
CDWRITE
using the command, 72
CE, 74
ceget, 211
ceinqr, 211
ceput, 212
cesel, 212
Character string
converting to integer string, 60
Characters retrieved from a double precision variable, 259
CMBLOCK, 75
cmopt common block, 185
cmopt.inc file, 104
cnvget subroutine, 145
/COM, 72
COMBIN7 and COMBIN37 parameters, 150
commands
creating, 174
common blocks, 104
cmopt, 185
component entities
defining, 75
constitutive material models
defining, 75
Constraint equation
defining constant term in, 74
Constraint equations
deleting or selecting, 212
getting, 211
getting information about, 211
storing, 212
Constraints
deleting at a node, 213
getting from the database, 213–214
getting information, 212
storing at a node, 213, 235
Contact
defining, 77
convection loads
defining, 145
converting nodal data to fixed format data blocks, 112
Coordinate system
deleting a coordinate system, 209
getting a coordinate system, 208
getting information about a coordinate system, 208
storing a coordinate system, 209
Coordinate Systems
defining, 79
Coupled nodes
defining, 75
Coupled set
getting a coupled set, 210
getting information about, 209
selecting or deleting, 211
storing a coupled set, 210
Coupling and constraint routines, 209
CP, 75
cpget, 210
cpinqr, 209
cpput, 210
cpsel, 211
creep1 subroutine, 139
creep3 subroutine, 139
csydel, 209
csyget, 208
csyiqr, 208
csyput, 209
CURVE
defining, 78
custom failure criteria, 168
custom optimization routine, 185
D
Database access routines, 195
Database and mode information
reading into ANSYS, 71
decks, 104
Degree of freedom labels
customizing, 73
Demonstration routines, 61
/DFLAB, 73
disdel, 213
disget, 212–213
displacement and coordinate debugging, 110
displacement values, 181
displacement vector debugging, 109
disput, 213
DOF pointers, 104
Double precision vectors, 265
dptoch subroutine, 259
dspdel, 235
dspget, 235
dspiqr, 234
dspput, 235
E
eccdel, 256
ecdget, 255
ecdiqr, 255
ecdput, 255
ehcpprm.inc file, 104
echrtr, 203
ecrdel, 248
ecreget, 246
ecriqr, 246
ecreput, 247
ecvdel, 224
ecvget, 223
ecviqr, 222
ecvput, 223
EDCADAPT, 76
edcdel, 229
EDCGEN, 77
edcget, 229
edciqr, 228
edcput, 229
EDCURVE, 78
EDCURVE, 78
EDDRELAX, 78
EDLCS, 79
EDLOAD, 79
EDPREAD, 80
EDWELD, 81
eeldel, 244
eelget, 243
eeliqr, 242
eelput, 243
efsdel, 233
efsget, 233
Element calculated heat generation
- deleting: 258
- getting: 258
- getting information about: 257
- storing: 258

Element convection
- deleting: 224
- getting: 223
- getting information about: 222
- storing: 223

Element current densities
- deleting: 256
- getting: 255
- getting information about: 255
- storing: 255

Element current density
- deleting: 229
- getting: 229

Element Energy
- deleting: 241
- getting: 240
- getting information about: 240
- storing: 241

Element euler angle
- deleting: 251
- getting: 250
- storing: 251

Element euler angles
- getting information about: 250

Element face flagged surface
- deleting: 233
- getting: 233
- getting information about: 234
- storing: 234

Element Face Impedance
- getting: 231

Element face pressure
- getting: 221
- storing: 222

Element fluence
- deleting: 228
- getting: 228
- getting information about: 227
- storing: 228

Element fluxes
- deleting: 252
- getting information about: 251
- storing: 252

Element Fluxes
- getting: 252

Element heat generation
- deleting: 227
- getting: 226
- getting information about: 226
- storing: 227

Element impedance
- deleting: 232
- getting information about: 231
- storing: 231

Element loading routines
- deleting: 231
- storing: 231

Element local force
- deleting: 253
- getting: 253
- storing: 253

Element local forces
getting information about, 252
element matrices, 128
accessing, 151
reusing, 133
Element miscellaneous non-summable data
deleting, 255
getting, 254
getting information about, 254
storing, 254
Element miscellaneous summable data
deleting, 237
getting, 236
getting information about, 236
storing, 236
Element nodal creep strain
deleting, 248
getting, 246
getting information about, 246
storing, 247
Element nodal elastic strain
deleting, 244
getting, 243
getting information about, 242
storing, 243
Element nodal force
getting, 237
getting information about, 237
storing, 237
Element nodal forces
deleting, 238
Element nodal gradients
deleting, 242
getting, 242
getting information about, 241
storing, 242
Element nodal plastic strain
deleting, 246
getting, 245
getting information about, 244
storing, 245
Element nodal stress
deleting, 240
getting, 238
getting information about, 238
storing, 239
Element nodal thermal strain
getting information about, 248
Element nodal thermal stress
getting, 248
storing, 249
Element nonlinear tables
deleting, 257
getting, 256
getting information about, 256
storing, 257
Element pressure/convection
deleting, 222
getting information about, 221
Element surface stress data
deleting, 233
getting, 232
getting information about, 232
storing, 233
Element temperature
deleting, 225
getting, 225
getting information about, 224
storing, 225
Element thermal, initial, and swelling strain
deleting, 250
Element virtual displacement
going, 230
getting information about, 230
storing, 230
Elements
 defining, 82
 example of element attribute routines, 209
 getting information about an element, 200
 getting information about an element type, 202–203
 getting information about element characteristics, 203
 getting the number of the next defined element, 197
 getting the number of the next element, 197
 getting the number of the previous selected element, 197
 routines for selecting and retrieving, 195
 selecting, unselecting, deleting, or inverting an element, 197
 selecting, unselecting, deleting, or inverting an element type, 204
 storing an element, 202
 storing element type data, 203
 elements, 101
 accessing information, 130
 changing face
 charge density surface information, 173
 convection surface information, 171
 heat flux surface information, 172
 changing pressure information, 171
 debugging, 110
 evaluating material properties, 135
 finding coordinates, 183
 load vectors, 135
 matrix debugging, 110
modifying, 146
   orientation of material properties and stress of layers, 149
monitoring, 146
non-summable miscellaneous data
   writing to results file, 143
overriding characteristic defaults, 128
parameters, 104
returning reference names, 131
rotation vectors
   computing, 143
rotational pseudovector
   updating, 144
storing
   user-supplied output, 148
subroutines, 103
   for element characteristics, 127
   for element creation, 131
transformation matrix
   computing, 143
types, 126
updating
   3-D nodal coordinates, 144
   creep history, 139
   plastic history, 137
   swelling history, 140
updating rotational pseudovector, 144
user-defined: creating, 113
writing saved variable set, 132
ElemGetMat subroutine, 123
efname, 253
efnget, 253
efniqr, 252
efnput, 253
EllInterp subroutine, 183
eLlenPsvrBuf function, 141
eLmiqr, 200
eLmput, 202
eLnest, 197
eLnxdf, 197
eLparm.inc file, 104
eLprev, 197
ElResultGet subroutine, 182
ElResultStrt subroutine, 182
eLsel, 197
eLucom.inc file, 104
eMndel, 255
eMnget, 254
eMiqr, 254
emnp, 254
Empdel, 237
Empget, 236
enSget, 236
EN, 82
enfdel, 238
enfget, 237
enfiqr, 237
enfput, 237
enget, 241
enfdel, 240
ens, 238
ensiq, 238
ensget, 239
ensput, 239
expdl, 245
expget, 245
expriq, 244
exprr, 222
enlget, 256
enlget, 257
ensdel, 240
ensget, 238
ensiq, 238
ensput, 239
epdl, 246
epget, 245
epliq, 244
eprr, 222
erhandler subroutine, 262
eriq subroutine, 260
Errors
   displaying, 262
   obtaining information, 260
etddf, 233
etdfget, 232
etfriq, 232
etfput, 233
ethdel, 250
ethet, 248
etfriq, 248
ethget, 249
etrdel, 222
etrget, 221
etqriq, 221
etrdel, 225
etrget, 225
etpqriq, 224
etpput, 225
etypcom-inc file, 104
etypget, 203
etypiq, 202
etypset, 203
etysel, 204
euidel, 251
eulget, 250
eulq, 250

Programmer's Manual for ANSYS. ANSYS Release 11.0. © SAS IP, Inc. All rights reserved.
Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.
Index

eulput, 251
evaluating material properties for 1-D elements, 137
evdget, 230
evdqiq, 230
evdput, 230
exinc4 subroutine, 60

z-buffered graphics, 89

H
heat generation
defining loads, 146
hgnget subroutine, 146
/HOLD command, 192
hygrothermal growth, 167
hyperelastic material models
creating, 165

I
I/O systems, 55
imove function, 265
impcom.inc file, 104
files
 .inc, 104
 .inc extension, 104
include files, 104
inexc4 subroutine, 60
Initializing paging space, 57
/INPUT, 72
Integer string
converting to character string, 60
Interpolation, 263
intrp subroutine, 263
izero function, 265

J
Jobname.cdb
defining, 80

L
large data files, 112
largeIntGet subroutine, 61
lastv function, 265
Libraries, 53
linking UPFs to ANSYS, 101, 105
load data
getting from analysis results, 182
reading for a substructure generation run, 134
load steps
accessing program logic, 183
containing information about, 104
load vectors, 128
accessing, 151
Loads
defining, 79
loads, 101
computing complex load vector, 147
convection, 145
customizing, 170

G
Geometry
relaxing, 78
getnod, 199
GetRForce function, 181
GetStackDisp function, 181
/FILE, 89
/GPRINT, 72
Graphics files
modifying, 89
neutral graphics file format, 90
pixmap format for graphic display files, 89

File access routines
abbreviations, 55
file headers, 111
File printing
adding comments to a file, 72
suppressing comments, 72
File status
retrieving, 55
Files
ansyscust, 106
binary
accessing, 53
characteristics of, 54
files, 105–106
START.ANS, 174
/FILNAM, 72
fordel, 215
forget, 214
foriqr, 214
forput, 214
FORTRAN, 101–103
Full transposed matrix, 270
functions
miscellaneous, 192
Functions
vector, 264

exinc4 subroutine, 60
debugging, 110
heat generation, 146
pressure, 145
temperature, 144
LOCAL, 82
Local coordinate system
defining, 82

M
M, 83
maat subroutine, 271
Master degree of freedom
defining, 83
material properties, 101
checking user-defined data, 167
customizing, 157
debugging, 110
modifying orientation of, 149
nonlinear information, 141
updating nonlinear strain history for, 166
Material properties
deleting a material property table, 206
getting a material property table, 205
getting information about a material property, 204
storing a material property table, 206
Material property data table
defining, 83
matrix
debugging, 109
Matrix
changing a matrix value, 271
filling upper triangle from lower triangle, 272
multiplying a vector by, 270
multiplying a vector by full transposed, 270
subroutines, 270
transposing, 271
transposing symmetric, 272
matsym subroutine, 272
matxb subroutine, 271
matvx subroutine, 270
matvx1 subroutine, 271
maxv subroutine, 270
maxv1 subroutine, 270
mctac subroutine, 272
memory management routines, 188
Meshing
defining, 76
meshing debugging, 111
miscellaneous functions, 192
monitoring elements, 146
Moving one vector into another, 266
MPDATA, 83
mpdel, 206
mpget, 205
mpinqr, 204
mpput, 206
MPTEMP, 83
mreuse subroutine, 133
multifield debugging, 109

N
N, 83
ndcdei, 219
ndcget, 219
ndcinqr, 219
ndcput, 219
ndgall, 199
ndgxyz function, 269
ndinqr, 198
ndnext, 195
ndnxdf, 196
ndprev, 196
ndpxyz subroutine, 269
ndsel, 196
ndspgt, 200
Neutral graphics file
character set used in, 90
color specification for, 93
decoding, 94
dependencies of, 94
format, 90
general format of, 90
plot directives, 91
Area-Color, 92
Draw, 92
End-Plot, 93
Graph-Color, 92
Line-Type, 92
Line-Width, 92
Marker Size, 92
Move, 92
No-Opt, 93
Normal, 93
parameter types for directives, 91
Pixmap, 93
Point, 92
Polygon, 93
Start_Plot, 92
Text, 92
Text-Color, 92
Text-Size, 92
Window, 92
Newton-Raphson debugging, 109
nfudei, 218
Index

nfuget F, 218
del, 218
nfuiqr, 217
get, 217
nfuput, 218
nfuget, 217
I
nhgdel, 217
nhgget, 217
hhgiqr, 216
nhgput, 217
nlget function, 141
nminfo subroutine, 131
nmisc record
storing data in, 142
nodel coordinate system transformation debugging, 111
Nodal current density
deleting, 219
getting, 219
getting information about, 219
storing, 219
nodel data conversion to fixed format data blocks, 112
Nodal fluence
deleting, 218
Nodal fluences
getting, 218
getting information about, 217
storing, 218
Nodal heat generation
deleting, 217
getting, 217
getting information, 216
storing, 217
Nodal load
deleting at a node, 215
storing at a node, 214
Nodal loading routines, 212
Nodal loads
getting information about, 214
Nodal magnetic virtual displacement
getting, 220
getting information about, 220
nodel orientation matrix
adjusting, 130
nodel reaction force values
getting, 181
Nodal results
getting from the database, 235
getting information about, 234
Nodal temperature
deleting, 216
getting, 216
getting information about, 215
storing, 216
Nodal virtual displacement
deleting, 221
storing, 220
Node information routines, 198
Nodes
defining, 83
getting a nodal point, 199
getting information about a node, 198
getting the next node number, 195
getting the nodal solution for a node of an element, 200
getting the number of the next defined node, 196
getting the number of the previous selected node, 196
getting the xyz/rotation coordinates vector for a node, 199
getting XYZ vector for, 269
routines for selecting and retrieving, 195
selecting, unselecting, deleting, or inverting a node, 196
storing a node, 199
storing XYZ vector for, 269
nonlinear debugging, 110
nonlinear strain history for materials, 166
/NOPRINT, 72
ntpdel, 216
ntpget, 216
ntpiqr, 215
ntpput, 216
nvddel, 221
nvdget, 220
nvdiqr, 220
nvdpout, 220
O
.o extension, 105
.obj extension, 106
object files, 105–106
optimization program
creating, 184
linking to ANSYS, 185
optimization variables, 104
optimizing ANSYS file reads, 112
orientation of material properties, 149
outpcm.inc file, 104
output
generating, 112
Output
obtaining information about, 259
output control, 104
P
parallel processing and UPFs, 102
Parameters
  retrieving system-dependent parameters, 56
parameters
  adding, 190
  ANSYS, 104
  creating a dimensioned parameter, 190
  defining in subroutines, 127
  element characteristics, 104
  finding and evaluating, 190
  performing user operations on COMBIN7 and COMBIN37, 150
  routines to process, 190
pardef subroutine, 190
pardim subroutine, 190
plast1 subroutine, 137
plast3 subroutine, 138
plasticity laws
  creating, 160
pointers to element data array, 104
POST1 postprocessor, 127
pressure loads
  defining, 145
printer output, 129
Printing file contents, 62
program logic
  accessing, 183
  tracking, 109
programmer debugging, 109
programming UPFs, 103
prope1 subroutine, 136
propev subroutine, 135
prsget subroutine, 145
pstev1 subroutine, 137
putnod, 199

R
  R, 85
  rdfull, 63
  rdsubs subroutine, 63
Reading a results file, 64
Reading an ANSYS substructure file, 63
Reading and reformatting the .FULL file, 63
reading large data files rapidly, 112
Real constant set
  defining, 85
real constants
  debugging, 110
  fetching, 135
Real constants
  getting information about a real constant set, 206
  getting real constant data, 207
  selecting or deleting a real constant set, 207
ResRdBegin function, 65
ResRdCsqs function, 67
ResRdDemo, 64
ResRdDisp function, 68
ResRdElem function, 67
ResRdEstr function, 69
ResRdFix function, 68
ResRdForc function, 69
ResRdGeomBegin function, 66
ResRdNode function, 67
ResRdReal function, 66
ResRdRfor function, 68
ResRdSolBeginI function, 68
ResRdType function, 66
restrictions on UPF creation, 101
Results
  deleting at a node, 235
Results file
  opening the file and retrieving global information, 65
  retrieving applied nodal constraints, 68
  retrieving applied nodal loads solution, 69
  retrieving coordinate systems, 67
  retrieving element solutions, 69
  retrieving element types, 66
  retrieving elements, 67
  retrieving geometry information, 66
  retrieving nodal coordinates, 67
  retrieving nodal solution, 68
  retrieving reaction solution, 68
  retrieving real constants, 66
  retrieving result set location, 68
Results files
  retrieving information from, 64
Results information routines, 234
results values
  getting at selected points, 182
ResWrDemo, 64
Retrieving
  maximum vector value for a location, 265
  position of last nonzero term in a vector, 265
reusing element matrices, 133
RLBLOCK, 85
rlget, 207
rlinqr, 206
rlsel, 207
rotr subroutine, 143
RunCommand function, 192
rvrget subroutine, 135

S
  saved variables
### Index

- **debugging**, 110
- **fetching**, 132
- **Scalar constants**
  - assigning to vectors, 266
- **scalar fields to user-defined values**, 170
- **SECBLOCK**, 85
- **SFBEAM**, 86
- **SFE**, 87
- **Simultaneous linear equations**, 273
- **solution control debugging**, 109
- **solutions**
  - accessing program logic, 183
  - debugging, 109
- **soptcm.inc file**, 104
- **source files**, 105
- **stack storage definition**, 104
- **stack.inc file**, 104
- **Standard ANSYS file header**, 57–58
- **START.ANS file**, 174
- **stepcm.inc file**, 104
- **Stopping ANSYS**, 264
- **storing**
  - user-supplied element output, 148
- **subrd subroutine**, 134
- **Subroutines**
  - accessing results files, 64
  - matrix, 270
  - tracking calls to, 261
- **subroutines**
  - for elements, 103
  - for memory management, 188
  - for processing parameters, 190
  - running ANSYS as, 174
  - supporting for user-created elements, 131
  - supporting user-defined commands, 181
- **subwrt subroutine**, 135
- **Surface load**
  - defining, 87
  - defining as beams, 86
- **svgidx subroutine**, 131
- **svpidx subroutine**, 133
- **svrget subroutine**, 132
- **svrput subroutine**, 132
- **swell1 subroutine**, 140
- **swell3 subroutine**, 140
- **swelling laws**, 164
- **symeqn subroutine**, 273
- **sysiqr function**, 55
- **system dependencies**, 103
- **systop subroutine**, 264

### T
- **table data**
  - retrieving, 137
- **tbuser subroutine**, 137
- **temperature loads**
  - defining, 144
- **Temperature table**
  - defining, 83
- **/TITLE**, 72
- **tmpget subroutine**, 144
- **TrackBegin subroutine**, 109, 261
- **TrackEnd subroutine**, 109, 262
- **tran subroutine**, 272
- **transient debugging**, 109
- **Transposing a matrix**, 271
- **trrot subroutine**, 143

### U
- **UAnBeg subroutine**, 183
- **UAnFin subroutine**, 183
- **uec100 routine**, 127
- **uec100 subroutine**, 127
- **uel100 subroutine**, 127–128
- **uel101 subroutine**, 103
- **UEIMatx subroutine**, 151
- **uelp100 subroutine**, 127, 129
- **uex100 subroutine**, 127–128
- **UltBeg subroutine**, 183
- **UltFin subroutine**, 183
- **ULdBeg subroutine**, 183
- **ULdFin subroutine**, 183
- **/UNDO command**, 192
- **UNIX systems**
  - compiling and linking UPFs on, 105
  - FORTRAN programming on, 103
- UPFs (see user programmable functions (UPFs))
  - unslly subroutine, 149
  - user programmable features (UPFs), 101
    - activating, 107
    - and loads, 101
    - and material properties, 101
    - before you begin using, 102
    - capabilities of, 101
    - compiling and linking
      - on UNIX systems, 105
      - on Windows systems, 106
    - considerations for using, 102
    - debugging, 109
    - definition of, 101
    - development strategy, 103
    - include files, 104
linking subroutines to user routines, 105
planning, 103
running your executable, 108
understanding, 103
verifying your routines, 108

user subroutines
for creating new elements, 113, 126
User subroutines, 127
user-defined commands, 174
making available to all ANSYS sessions, 174
user-defined elements, 113
User-defined variable storage, 104
user-programmable features
user-defined elements, 113
user01 command, 174–175
user02 command, 176
user03 command, 177
USER300 user-defined element, 114
userac, 209
userac subroutine, 127, 130
useran subroutine, 149
userch subroutine, 173
userck subroutine, 167
usercr subroutine, 161
usercreep subroutine, 161
usercv subroutine, 171
UserElem subroutine, 115
userer subroutine, 142
userfd subroutine, 147
userfx subroutine, 172
UserHyper subroutine, 165
usermat subroutine, 157
usermc subroutine, 167
userop subroutine, 101, 185
userou subroutine, 148
userpe function, 150
userpl subroutine, 160
userpr subroutine, 171
userrc subroutine, 150
usersw subroutine, 164
usertr subroutine, 127, 130
UserVisLaw subroutine, 168
uservp subroutine, 166
USsolBeg subroutine, 183
USsolFin subroutine, 183
USRCAL command, 170, 183
usrefl subroutine, 170
usrfc6 subroutine, 168
UsrFictive subroutine, 152
usrsurf116 subroutine, 154
UsrVisCE subroutine, 153
USSSbeg subroutine, 183
USSSFin subroutine, 183
USTRESS subroutine, 152
usvrcm.inc file, 104
UTHICK subroutine, 151

V
vamb subroutine, 267
vamb1 subroutine, 267
vapb subroutine, 267
vapb1 subroutine, 267
vapcb1 subroutine, 267
variables
defining additional, 111
fetching the index for saved, 131
optimization, 104
writing saved element index, 133
Variables
retrieving characters from, 259
vcross subroutine, 268
vdot function, 264
Vector functions, 264
Vectors
assigning equal values to two vectors, 265
assigning scalar constants to, 266
assigning scalar constants to incrementally, 266
combining two in one, 267
defining by a cross product, 268
getting the difference of two, 267
getting XYZ vector for a node, 269
initializing to zero, 265
moving one into another, 266
moving one into another incrementally, 266
multiplying a matrix by, 270
multiplying by a full transposed matrix, 271
multiplying one by a constant, 268
multiplying to a constant, 267
normalizing a three-component vector, 268
normalizing to unit length, 269
setting integer vector to zero, 265
setting one to be the sum of two vectors, 267
subtracting one from another, 267
velocity information, 104
viinit subroutine, 266
vimove subroutine, 266
vinit subroutine, 266
viscoplastic/creep material behavior, 161
viscosity laws
defining, 168
vmax function, 265
vmult subroutine, 268
Index

vmult1 subroutine, 268
vnorm subroutine, 269
vnorme subroutine, 268
vsum function, 264
vzero subroutine, 265

W
wavefront reordering debugging, 111
Weld
defining, 81
Windows systems
  compiling and linking UPFs on, 106
wrinqr function, 259
Writing an ANSYS substructure file, 63
wrtsub subroutine, 63

X
xyzup3 subroutine, 144

Z
Z-buffered graphics, 89